TESIS-KI142502

Pencarian Kode Sumber Perangkat Lunak pada Media Sosial StackOverflow Menggunakan Lokasi Konsep

ACHMAD ARWAN
NRP 5112201010

DOSEN PEMBIMBING
Dr. Ir. Siti Rochimah, MT.
Rizky Januar Akbar, S.Kom, M.Eng

PROGRAM MAGISTER
BIDANG KEAHLIAN REKAYASA PERANGKAT LUNAK
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2015

THESES-KI142502

Code Search on StackOverflow Using Concept Location

ACHMAD ARWAN
NRP 5112201010

SUPERVISOR
Dr. Ir. Siti Rochimah, MT.
Rizky Januar Akbar, S.Kom, M.Eng

MASTER PROGRAM
SOFTWARE ENGINEERING
DEPARTMENT OF INFORMATIC ENGINEERING
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2015

LEMBAR PENGESAHAN TESIS

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M.Kom.)
di Institut Teknologi Sepuluh Nopember Surabaya

oleh :

ACHMAD ARWAN

NRP. 5112201010

Dengan Judul:

Pencarian Kode Sumber Perangkat Lunak pada Media Sosial StackOverflow Menggunakan Lokasi Konsep

Tanggal Ujian : 5 Mei 2015

Periode Wisuda : 2015 Genap

Disetujui oleh :

1. Dr. Ir. Siti Rochimah, MT
   NIP. 196810021994032001 ..... (Pembimbing I)
2. Rizky Januar Akbar, S.Kom., M.Eng.
   NIP. 198701032014041001 ..... (Pembimbing II)
3. Daniel O. Siahaan, S.Kom, M.Sc, P.DEng.
   NIP. 197411232006041001 ..... (Penguji I)
4. Sarwosri, S.Kom, MT.
   NIP. 197608092001122001 ..... (Penguji II)
5. Ridho Rahman Hariadi, S.Kom, M.Sc
   NIP. 198702132014041001 ..... (Penguji III)

Direktur Program Pasca Sarjana,

Prof. Dr. Ir. Adi Soeprijanto, M.T.
NIP. 196404051990021001

PENCARIAN KODE SUMBER PERANGKAT LUNAK PADA MEDIA SOSIAL STACKOVERFLOW MENGGUNAKAN LOKASI KONSEP

Nama Mahasiswa : Achmad Arwan
NRP : 5112201010
Pembimbing : Dr. Ir. Siti Rochimah, MT
Rizky Januar Akbar, S.Kom, M.eng

ABSTRAK

Pencarian kode sumber di internet merupakan area penelitian yang cukup populer. Berbagai pendekatan telah dilakukan untuk mendapatkan kode sumber di internet. StackOverflow menyediakan sarana bagi para pemrogram untuk tanya jawab seputar kode. Pencarian kode di StackOverflow pada penelitian sebelumnya menggunakan metode tf-idf yaitu mencari berdasarkan jumlah kemunculan kata. Penelitian tersebut memiliki kelemahan yaitu penamaan variabel atau fungsi dianggap kata biasa, padahal variabel atau metode seringkali merupakan gabungan dua kata atau lebih. Sebagai contoh ada fungsi bernama *randomString* maka yang disimpan juga kata *randomString.* Sehingga jika mencari kata *random* maka kode yang ada pada fungsi *randomString* kemungkinan tidak akan direkomendasikan karena kata *randomString* dan *random* merupakan kata yang berbeda begitu juga sebaliknya. Permasalahan tersebut dapat diatasi dengan lokasi konsep. Lokasi konsep telah lama digunakan untuk mendapatkan korelasi antara kode dengan fitur atau konsep tertentu. Penelitian terdahulu tentang lokasi konsep berfokus pada kode dan komentar serta relasinya.

Penelitian ini mengajukan sebuah mekanisme mencari kode pada StackOverflow menggunakan lokasi konsep. Semua pertanyaan, jawaban, & kode yang berlabel Java diunduh dari server StackOverflow. Data kemudian diekstrak menjadi *corpus.* Lokasi konsep pada *corpus* dicari dengan algoritma LDA (*Latent Dirichlet Allocation)*. Pengguna cukup memasukkan konsep yang akan dicari ke dalam sistem. Sistem kemudian memberikan rekomendasi kode yang relevan dengan lokasi konsep yang diinputkan. Dengan mekanisme tersebut diatas maka pemrogram mendapatkan rekomendasi kode dengan mudah. Hasil penelitian ini mencapai hasil dengan tingkat *precision 0.52%* dan *recall 0.72%.*

Kata Kunci : pencarian kode sumber, StackOverflow, lokasi konsep, *corpus, latent dirichlet allocation*

CODE SEARCH ON STACKOVERFLOW USING CONCEPT LOCATION

Student Name : Achmad Arwan
Student Identity Number : 5112201010
Suvervisor : Dr. Ir. Siti Rochimah, MT
Rizky Januar Akbar, S.Kom, M.eng

ABSTRACT

Internet code search is quite popular research area. Various approaches have been made to get the source code from the internet. StackOverflow allows programmers to ask and answer questions about code. Previous approach to search code on StackOverflow use tf-idf method which based on number of occurrences of the word to recommend the codes. This method has the disadvantage that naming a variable or function are considered as normal words, even though a variable or function are often a combination of two or more words. For example if there is a method named “randomString” then stored well wording randomString. So if we search using keyword “random” the system probably will not recommend “randomString” because both words are different. Concept location may tackle this problem. Concept location has been used widely to obtain the correlation between code with a specific concepts or features. Previous research of concept location only focused on codes comments, and relation among the objects within the codes.

This research proposes a mechanism for finding source code on StackOverflow using concept location. Questions, answers & source code about Java code are downloaded from StackOverflow to local repository. Corpuses are generated by extracting questions, answers and code snippets. Inferencing concept location of codes created using LDA(Latent Dirichlet Allocation) algorithm. Users querying concepts to system and then system will recomend the codes based on relevant concepts. By using this mechanism, programmers are able to get the recommendations code easily. Results of this study achieve results with a level of precision and recall of 0.52% to 0.72%.

Keywords : code search, StackOverflow, concept location, corpus, latent dirichlet allocation

KATA PENGANTAR

Puji syukur kepada Allah SWT atas segala berkah dan anugerah-Nya sehingga penulis dapat menyelesaian tesis yang berjudul "Pencarian Kode Sumber Perangkat Lunak Pada Media Sosial Stackoverflow Menggunakan Lokasi Konsep".

Tesis ini disusun untuk memenuhi sebagian persyaratan untuk memperoleh gelar Magister Komputer di Institut Teknologi Sepuluh Nopember Surabaya. Tesis ini dapat terselesaikan tidak lepas dari bantuan dan dorongan yang sangat berharga dari berbagai pihak. Oleh sebab itu penulis mengucapkan terimakasih dan penghargaan yang sebesar-besarnya kepada berbagai pihak sebagai berikut.

1. Ibu Dr. Ir. Siti Rochimah, MT sebagai pembimbing I yang telah banyak meluangkan waktu dan pikiran dalam membimbing penulis untuk menyelesaikan tesis ini.
2. Bapak Rizky Januar Akbar, S.Kom., M.Eng. selaku pembimbing II yang telah membantu dalam penyusunan tesis dan karya tulis ilmiah.
3. Semua pihak yang telah membantu penulis dalam menyelesaikan tesis ini

Penulis menyadari bahwa dalam laporan tesis ini masih banyak kekurangan, oleh karena itu masukan dan saran yang membangun demi perbaikan dan pengembangan tesis ini sangat penulis harapkan. Akhir kata semoga tesis ini bermanfaat bagi para pembaca dalam pengembangan ilmu pengetahuan di negeri ini.

Surabaya, 13 Mei 2015

Penulis

DAFTAR ISI

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Dalam membuat sebuah program aplikasi, seorang pemrogram sering kali dihadapkan dalam permasalahan pengodean yang baru. Permasalahan ini solusinya mungkin bisa hanya dengan kode, komponen atau bahkan serangkaian komponen yang dapat digunakan untuk menyelesaikan permasalahan tersebut. Permasalahan-permasalahan tersebut bukan tidak mungkin juga telah diselesaikan oleh pemrogram di belahan dunia yang lain. Pemrogram dapat memanfaatkan situs forum tanya jawab seputar kode yaitu StackOverflow untuk mencari kode sebagai solusi atas permasalahannya. Cara kerja dari StackOverflow yaitu pengguna menulis pertanyaan tentang permasalahan yang dihadapi, kemudian pengguna yang lain menjawab pertanyaan tersebut. Penanya dapat memverifikasi kebenaran jawaban tersebut dengan memilih jawaban yang benar dari kesekian jawaban yang ada. Biasanya sebelum menulis pertanyaan, pengguna dapat mencari melalui form pencarian dengan kata kunci dari pertanyaan mereka. StackOverflow kemudian menampilkan beberapa pertanyaan dan jawaban yang mendekati dengan kata kunci tersebut. Mencari kode yang relevan tentunya tidak mudah, karena seringkali pemrogram harus mengerti dengan kode orang lain tersebut. Perlu pengujian dan penyesuaian dalam menggunakan kode yang direkomendasikan sebab jika tidak justru dapat menimbulkan masalah baru.

Beberapa penelitian telah dilakukan untuk mengatasi permasalahan pencarian kode di internet. Salah satu penelitian tersebut dilakukan oleh Ossher dkk (Ossher, 2009). Ossher membuat sebuah database yaitu SourcererDB. Database ini merupakan kumpulan kode Java yang diambil dari situs penyedia penyimpanan kode seperti SourceForge, Apache & Java.net. Kode-kode tersebut dibaca kemudian dimasukkan ke dalam entitas-entitas yang sesuai dengan deklarasinya. Jika kelas maka dimasukkan kedalam entitas kelas, metode dimasukkan ke entitas metode dsb. Pengguna cukup memasukkan kata kunci yang akan dicari, kemudian situs akan mencari dalam database dan memberikan rekomendasi kode yang

relevan dengan kata kunci tersebut. Tidak ada perbandingan hasil yang disampaikan dalam penelitian tersebut.

Penelitian yang paling dekat dengan penelitian ini adalah penelitian Zagalsky dkk (Zagalsky, 2012). Mereka menggunakan media sosial tanya jawab StackOverflow untuk memberikan rekomendasi kode. Dalam situs StackOverflow tersedia pertanyaan, jawaban yang berisi langkah dan potongan kode sumber. Zagalsky dkk mengambil semua pertanyaan yang berkaitan dengan jQuery dan memiliki jawaban yang terverifikasi untuk kemudian di simpan dalam repositori. Kemudian dengan metode tf-idf mereka kemudian melakukan perangkingan dokumen terhadap pertanyaan yang diajukan. Mereka merekomendasikan kode yaitu dengan cara memberi bobot tertentu pada judul, label, pertanyaan, jawaban, dan kode untuk kemudian dikalikan dengan nilai dari kemunculan kata pada dokumen. Bobot yang paling besar terletak pada judul dan label yaitu 4 dari skala 1 sampai 4. Sementara pertanyaan berbobot 2 dan jawaban kode hanya berbobot 1. Bobot tersebut digunakan untuk memberikan prioritas pencocokan kata. Sehingga judul dan label lebih prioritas dari pertanyaan jawaban dan kode. Mereka kemudian mempublikasikan melalui situs Exampleoverflow.net untuk mempermudah akses pengguna. Pengujian pada penelitian tersebut berdasarkan jumlah hasil queri bila dibandingkan dengan kakas lain seperti Google Search, Krugle, Koders dan StackOverflow sendiri. Selain itu pengujian lainnya menggunakan perbandingan pergantian konteks yaitu berapa jumlah klik yang diperlukan untuk melihat kode diantara kakas-kakas tersebut diatas.

Lokasi konsep dalam kode dapat digunakan untuk mencari kode yang relevan. Lokasi konsep menurut Biggerstaf adalah bagaimana mengidentifikasi lokasi awal dalam kode sumber yang berkaitan dengan fungsional khusus (Ted J. Biggerstaff, 1994). Menentukan lokasi konsep dalam kode dapat dilakukan dengan beberapa pendekatan. Salah satu pendekatan telah dilakukan oleh Adrian dkk, yaitu dengan membuat sistem temu kembali lokasi konsep pada kode dengan LSI(*Latent Semantic Indexing)* (Marcus, 2004). Savage dkk kemudian juga melakukan eksplorasi topik pada kode dengan metode LDA(*Latent Dirichlet Allocation)* dimana dalam serangkaian kode kemungkinan mengandung beberapa topik

(Savage, 2010). Keduanya mengekstrak kode untuk diambil nama variabel, metode, ataupun komentar untuk dibuat menjadi *Corpus.* Pertanyaan dalam StackOverflow sangat beragam. Allamanis dkk telah meneliti pertanyaan-pertanyaan apa dan kaitannya dengan kode dalam StackOverflow (Allamanis, 2013). Mereka menemukan beberapa variasi pertanyaan diantaranya yaitu pengguna sudah membuat kode tetapi kodenya tidak berjalan sesuai yang inginkan. Variasi lain yaitu pengguna tidak mengerti cara kerja sebagian kode tertentu. Selain itu ada pula yang bertanya bagaimana cara menggunakan potongan kode tersebut. Serta banyak variasi pertanyaan lain yang tidak dapat disebutkan satu persatu. Dari penelitian tersebut juga dapat diartikan bahwa kode dalam StackOverflow tidak hanya ada pada jawaban saja melainkan juga terdapat pada pertanyaan. Dari banyaknya variasi tersebut maka mencari kode yang relevan dengan konsep lokasi menjadi pilihan utama karena dengan konsep lokasi dapat diketahui maksud dari kode tersebut dimanapun letaknya, baik di pertanyaan atau jawaban.

Penelitian ini mengusulkan perbaikan dari penelitian Zagalsky dkk (Zagalsky, 2012). Pada penelitian sebelumnya, Zagalsky mencari kesamaan judul, label, pertanyaan, jawaban dan kode dengan *corpus* berdasarkan jumlah kemunculan kata. Zagalsky tidak memperhatikan pemecahan nama metode. Semisal ada sebuah fungsi bernama *randomString*, Zagalsky hanya akan mencatat kata *randomString* sebagai satu kesatuan utuh. Sehingga jika ada pertanyaan tentang *random* kemungkinan kode dalam fungsi *randomString* tidak masuk dalam rekomendasi selama tidak ada kata *random* pada judul, label, pertanyaan, dan jawaban. Sedangkan dalam lokasi konsep, pemecahan nama variabel atau fungsi merupakan hal utama dalam mendapatkan makna dalam sebuah kode. Jika menggunakan lokasi konsep, dalam fungsi *randomString* mengandung dua konsep yaitu *random* dan *string.* Sehingga penelitian ini menggunakan lokasi konsep untuk mendapatkan rekomendasi kode untuk meningkatkan akurasi pencarian.

Penelitian ini juga memperbaiki penelitian tentang lokasi konsep yang umumnya hanya berkutat pada kode dan komentar maka penelitian ini juga menggunakan fitur tambahan yaitu pertanyaan dan jawaban sebagai penjelas maksud kode yang ada pada pertanyaan atau jawaban.

Langkah dalam penelitian ini yaitu dengan mengunduh kemudian mengekstraksi pertanyaan, jawaban dan kode yang berlabel Java menjadi *corpus*. *Corpus* tersebut kemudian disimpan dalam repositori dan diolah menggunakan algoritma LDA. Pemrogram cukup memasukkan konsep yang ingin dicari. Kemudian diperoleh jawaban yang berisi kode atau komponen yang berisi konsep yang dimaksud dan selanjutnya bagian kodenya dipasang ke dalam kode pada IDE Eclipse.

1.2 Perumusan Masalah

Berikut ini adalah perumusan masalah pada penelitian ini. Bagaimana mencari kode yang sesuai dengan solusi permasalahan di media sosial StackOverflow dengan menggunakan konsep lokasi yang terdapat pada pertanyaan, jawaban dan kode?

1.3 Tujuan dan Manfaat Penulisan

Tujuan penelitian yang ingin dicapai adalah bagaimana mendapatkan kode bahasa Java yang relevan pada media sosial StackOverflow.

Manfaat yang diberikan adalah agar seorang pemrogram dapat informasi secara lengkap solusi dan kode yang digunakan untuk menyelesaikan permasalahannya.

Kontribusi yang dapat diberikan adalah membuat metode untuk mendapatkan kode yang relevan pada media sosial StackOverflow, sehingga memudahkan pemrogram dalam penggunaan kembali kode sumber.

1.4 Batasan Masalah

Dalam penelitian ini, pembahasan yang dicakup adalah sebagai berikut.

a. Menggunakan 153 data acak dalam topik *sort, database, file text, graphic & thread* dalam kategori *java* pada situs StackOverflow sebagai pusat pengetahuan permasalahan dan solusi.
b. Mendapatkan kode sumber berbasis Java yang relevan dari situs StackOverflow menggunakan konsep lokasi.

[Halaman ini sengaja dikosongkan]

BAB 2
KAJIAN PUSTAKA

2.1 Penelitian Terdahulu pada StackOverflow

StackOverflow adalah situs tanya jawab untuk para pemrogram yang antusias dan profesional (StackOverflow, 2014). Orang dapat mendaftar kemudian membuat pertanyaan dan menjawab pertanyaan dari pengguna lain. Beberapa fitur dalam situs ini selain tanya jawab antara lain adalah *Tags* yaitu kelompok label pada pertanyaan, *Badges* yaitu suatu hadiah yang diberikan oleh situs tersebut kepada penggunanya jika pengguna telah mencapai syarat yang telah ditentukan. Selain itu juga terdapat *Unanswered* yaitu pertanyaan yang belum terjawab. Pengguna dapat memvoting apakah pertanyaan atau jawaban tersebut berguna atau tidak. Selain itu penanya dapat memverifikasi jawaban tersebut apakah merupakan jawaban yang benar atau tidak. StackOverflow menyediakan repositori data yang dapat diunduh dan digunakan sesuai dengan keperluan. Data yang disediakan berupa file XML dari masing-masing tabel yang ada pada basis data StackOverflow. File yang digunakan adalah file "*Posts.xml"* yaitu file yang berisi semua posting baik pertanyaan maupun jawaban. Dalam file tersebut terdiri dari atribut-atribut yang berisi data dari posting. File tersebut kemudian ditransformasi ke dalam tabel pada basis data MySql. Berikut ini adalah struktur atribut dari tabel *Posts*.

Tabel 2. 1 Struktur Atribut tabel *Posts*

Nama Atribut	Tipe data	Kegunaan
ID	*Int*	Merupakan kode primer yang membedakan baris satu dengan baris yang lain.

Nama Atribut	Tipe data	Kegunaan
PostTypeId	*tinyint*	Merupakan kode untuk membedakan pertanyaan dan jawaban, jika 1 Pertanyaan dan jika 2 maka jawaban.
AcceptedAnswerId	*Int*	Yaitu kode posting dari jawaban atas posting baris ini, hanya ada jika baris ini bertipe 1(pertanyaan).
ParentId	*Int*	Merupakan kode induk dari jawaban baris ini, hanya jika baris ini bertipe 2 atau jawaban.
CreationDate	*Date*	Yaitu tanggal kapan posting ini dibuat.
Score	*Int*	Yaitu nilai dari posting ini pengguna lain yang memberikan rating.
ViewCount	*Int*	Menunjukkan berapa kali posting ini dilihat oleh pengguna.
Body	*Text*	Yaitu isi dari posting, disini berisi pertanyaan, jawaban, dan kode. Field ini berisi tag html sehingga harus dilakukan praproses agar menampilkan data yang sebenarnya.

Nama Atribut	Tipe data	Kegunaan
OwnerUserId	*Int*	Yaitu kode pengguna siapa yang membuat posting ini.
OwnerDisplayName	*Varchar*	Yaitu tampilan nama pengguna pembuat posting ini.
LastEditorUserId	*Int*	Yaitu kode pengguna yang melakukan pengeditan terakhir.
LastEditorDisplayName	*Varchar*	Yaitu nama pengguna yang melakukan editing terakhir.
LastEditDate	*Date*	Yaitu tanggal terakhir posting diubah.
LastActivityDate	*Date*	Yaitu tanggal terakhir aktifitas pada posting ini.
Title	*Varchar*	Yaitu judul dari posting ini.
Tags	*Text*	Yaitu tanda masuk kategori yang manakah posting ini.
AnswerCount	*Int*	Yaitu jumlah jawaban yang menjawab posting ini.
CommentCount	*Int*	Yaitu jumlah komentar yang mengomentari posting ini.
FavoriteCount	*Int*	Yaitu jumlah berapa orang yang memfavoritkan posting ini

Nama Atribut	Tipe data	Kegunaan
ClosedDate	*Date*	Yaitu tanggal kapan posting ini ditutup.

Dari semua atribut diatas, tidak semua atribut digunakan dalam penelitian ini. Atribut yang penting tersebut (*Id, Question, Answer)* dipilih oleh karena relevansinya dengan penelitian yang dilakukan. Atribut-atribut penting tersebut digabung menjadi sebuah tabel baru yang merupakan wadah dari proses-proses yang dilakukan dalam penelitian ini. Atribut-atribut baru digunakan untuk menyimpan hasil proses dari atribut yang sudah ada sebelumnya, hal ini diperlukan untuk memudahkan pengecekan apakah proses tersebut sudah benar. Berikut ini struktur tabel final yaitu tabel kompilasi dari proses pada penelitian ini.

Tabel 2. 2 Struktur Tabel Final

Nama Atribut	Tipe data	Penjelasan
Id	*Varchar*	Yaitu kode pembeda baris satu dengan baris yang lain
Question	*Text*	Yaitu berisi pertanyaan yang merupakan pemetaan dari atribut *body* yang diambil dari posting yang bertipe pertanyaan saja. Kode program kemungkinan juga terdapat dalam atribut ini.
Answer	*Text*	Yaitu berisi jawaban dari pertanyaan dimana juga merupakan pemetaan atribut body yang diambil dari posting yang bertipe jawaban. Kode program kemungkinan juga terdapat dalam atribut ini.

Nama Atribut	Tipe data	Penjelasan
Posting	*Text*	Yaitu gabungan dari pertanyaan dan jawaban yaitu field *question & answer.*
Text	*Text*	Yaitu atribut *Posting* yang kemudian dihilangkan *tag* html dan simbol lainnya sehingga hanya tinggal kata kata saja.
Splittedcamel	*Text*	Yaitu atribut untuk menyimpan hasil proses pemisahan penamaan variabel yang menggunakan aturan *camelCase & underscore(_)* yang ada pada atribut *text.*
Stopwordremoved	*Text*	Yaitu atribut untuk menyimpan hasil proses eliminasi kata *stopword* atribut *Splittedcamel.*
Stemmedwords	*Text*	Yaitu atribut untuk menyimpan hasil proses *stemming* pada atribut *Stopwordremoved.*

Beberapa peneliti telah menggunakan StackOverflow sebagai objek penelitian mereka. Zagalsky dkk (Zagalsky, 2012) membuat mekanisme untuk merekomendasikan kode pada situs tanya jawab StackOverflow. Mereka mengambil semua pertanyaan dan jawaban yang memiliki tanda jQuery untuk disimpan dalam repositori. Dari pertanyaan dan jawaban dibuat *Corpus* untuk kemudian dilakukan pengurutan dengan menggunakan bantuan Apache Lucene. Metode yang digunakan dalam penelitian tersebut yaitu tf-idf (*term frequency-inverse document frequency*). Cara kerjanya yaitu pengguna memasukkan kata kunci kemudian mesin mencari kata tersebut dalam kode dan metadata yang terkait. Mereka merekomendasikan kode yaitu dengan cara memberi bobot tertentu pada judul, label, pertanyaan, jawaban, dan kode untuk kemudian dikalikan dengan nilai

dari kemunculan kata pada dokumen. Bobot yang paling besar terletak pada judul dan label yaitu empat dari skala satu sampai empat. Sementara pertanyaan, jawaban berbobot dua dan kode hanya berbobot satu. Bobot tersebut digunakan untuk memberikan prioritas pencocokan kata. Sehingga judul dan label lebih prioritas dari pertanyaan jawaban dan kode. Mereka kemudian mempublikasikan melalui situs Exampleoverflow.net untuk mempermudah akses pengguna. Penelitian tersebut menggunakan pengukuran bagaimana hasil queri bila dibandingkan dengan kakas lain seperti google search, krugle, koders dan StackOverflow sendiri. Selain itu juga menggunakan perbandingan pergantian konteks yaitu berapa jumlah klik yang diperlukan untuk melihat kode dengan kakas bantu tersebut diatas. Cara ini hanya melihat kemunculan kode sebagai kata. Selain itu metode penamaan metode seperti metode *camelCase* yang biasanya terdiri dari dua kata atau lebih akan dianggap sebuah kata saja padahal mungkin terkandung lebih dari satu makna konsep dalam kode tersebut.

Penelitian lain yaitu tentang keragaman pertanyaan dalam StackOverflow. Allamanis dkk (Allamanis, 2013) telah meneliti pertanyaan-pertanyaan apa yang sulit diselesaikan dan kaitannya dengan kode dalam StackOverflow. Mereka menemukan beberapa variasi pertanyaan diantaranya yaitu pengguna sudah membuat kode tetapi kodenya tidak berjalan sesuai yang inginkan sehingga mereka butuh perbaikan tentang kode tersebut. Variasi lain yaitu pengguna tidak mengerti cara kerja sebagian kode tertentu. Selain itu ada pula yang bertanya bagaiamana cara menggunakan potongan kode tersebut. Serta banyak variasi pertanyaan lain yang tidak dapat disebutkan satu persatu. Dari penelitian tersebut juga dapat diartikan kode dalam StackOverflow tidak hanya ada pada jawaban saja melainkan juga terdapat pada pertanyaan. Selain itu mereka juga menggunakan lokasi konsep untuk mendapatkan topik dalam kode. Penelitian tersebut tidak merekomendasikan kode melainkan mencari isu apa yang paling rumit yang terkait dengan pengidentifikasi dalam kode. Tabel berikut menunjukkan penelitian terdahulu yang menggunakan data dari StackOverflow.

Tabel 2. 3 Beberapa Penelitian Terdahulu Menggunakan StackOverflow.

Nama	Deskripsi	Hasil	Kesimpulan
Zagalsky dkk	Memberi rekomendasi kode berdasarkan pencarian kata kunci dari repositori pertanyaan jawaban dan kode menggunakan metode *term frequency inverse document frequency*.	Paling banyak memberikan rekomendasi kode serta paling sedikit berpindah konteks dalam mencari kode diantara beberapa alat pencarian kode di internet lain.	Penelitian ini mampu memberi rekomendasi kode yang banyak serta memudahkan pemrogram dalam mencari kode walaupun masih sebatas pencarian berdasarkan kata-kata.
Allamanis dkk	Mencari pertanyaan-pertanyaan sulit yang mana yang berhubungan dengan pengidentifikasian variabel atau metode pada kode.	Menghasilkan hubungan antara teks dan topik-topik kode dengan jenis pertanyaan.	Penelitian ini menggambarkan variasi pertanyaan dalam StackOverflow & kaitannya dengan teks dan topik dalam kode.

Selain dua penelitian diatas masih banyak penelitian lain yang menggunakan StackOverflow sebagai objek penelitian, akan tetapi isinya kurang relevan dengan penelitian ini. Penelitian tersebut antara lain oleh Kartik dkk (Kartik Bajaj, 2014), Bogdan dkk (Bogdan Dit, 2011), Alexander dkk (Alexander Halavais, 2014), Squire dkk (Squire, 2014), Tausczik dkk (Tausczik, 2014).

2.2 Lokasi Konsep

Lokasi konsep menurut Ted J. Biggerstaf (Ted J. Biggerstaff, 1994) adalah bagaimana cara untuk mengidentifikasi lokasi awal dalam kode sumber yang

berkaitan dengan fungsional khusus tertentu. Sebagai contoh yaitu mencari dimana fungsi untuk melakukan faktorisasi dalam baris kode. Beberapa pendekatan telah dilakukan untuk menyelesaikan permasalahan dalam mencari lokasi konsep dalam kode sumber.

Marcus dkk (Marcus, 2004) mengajukan sebuah metode untuk mencari konsep lokasi pada kode berorientasi objek bahasa C secara statis. Mereka menggunakan metode LSI(*Latent Semantic Indexing)* untuk mencari lokasi konsep dalam kode. Caranya dengan membuat *corpus* dari ekstraksi kode dan komentar. Setelah terbentuk *corpus* kemudian pengguna memasukkan konsep yang akan dicari dan dengan LSI akan didapatkan *corpus* yang paling dekat dengan konsep yang dicari tersebut. Pengujian dilakukan dengan membandingkan dengan metode-metode lain yaitu grafik depedensi dan pencarian pola kata menggunakan GREP. Hasil penelitian ini lebih baik dari grafik depedensi karena dapat mendeteksi kode fungsi yang tidak terdeteksi oleh metode grafik depedensi. Selain itu pengguna dapat mencari lebih dari satu kata kunci sebagai input pencariannya.

Wilde dkk (Wilde, 2003) kemudian meneliti perbandingan metode untuk mencari fitur pada perangkat lunak warisan. Mereka menggunakan kode perangkat lunak yang dikembangkan sejak lama kemudian membandingkan metode Software Reconnaissance, grafik dependensi, serta Grep dalam mencari konsep lokasi. Berikut ini kelebihan dan kekurangan masing-masing metode menurut Wilde.

Tabel 2. 4 Kelebihan dan Kekurangan Metode dalam Mencari Lokasi Konsep (Wilde,2003).

Metode	Deskripsi	Kelebihan	Kekurangan
Software reconnaissance	Membandingkan jejak eksekusi kode antara yang menggunakan fitur dan tidak.	Cukup cepat, fokus pada potongan kode yang kecil.	Fitur harus dapat dikontrol melalui input data dan paling baik untuk kode lokal yang sudah dipahami.
Grafik dependensi	Pencarian kode secara sistematis	Fleksibel, memberikan	Memerlukan pencarian yang

	melalui grafik ketergantungan antara kode satu dengan kode lain.	pemahaman konteks fitur.	luas dan memakan banyak waktu serta lebih cocok untuk kode yang modular.
Grep	Pencarian teks dengan kata kunci yang ada pada komentar maupun variabel.	Sangat cepat dan banyak alat bantu yang bisa digunakan.	Kurang bisa diandalkan, dan memerlukan kata kunci petunjuk yang bagus dalam kode.

Savage dkk (Savage, 2010) meneliti bagaimana mendapatkan topik dalam kode menggunakan metode LDA (*Latent Dirichlet Allocation*). Objek penelitiannya pada kode sumber berbasis Java. Mereka mengekstrak kata dari kode kemudian diproses dan mendapatkan topik dari kode-kode sumber dengan menggunakan LDA. Kemudian dari hasil ekstraksi tersebut dibuat visualisasi untuk memetakan struktur kode berdasarkan konsepnya dan hubungan antara konsep tersebut. Pengguna juga dapat mencari konsep yang ingin dicari dan yang tampil adalah kelas yang relevan dengan konsep tersebut. Hasil penelitian ini berupa alat bantu dalam bentuk *plugin* dari IDE Eclipse bernama Topic_{xp}. Selain itu pemrogram dapat dengan mudah dan cepat dalam mencari konsep sehingga mempermudah proses perawatan perangkat lunak. Dari beberapa penelitian diatas maka dapat dilihat ringkasan penelitian tentang lokasi konsep pada tabel berikut ini.

Tabel 2. 5 Ringkasan Beberapa Penelitian Terdahulu Tentang Lokasi Konsep.

Nama	Deskripsi	Hasil	Kesimpulan
Marcus dkk	Mencari lokasi konsep menggunakan LSI pada kode sumber berbasis objek berbahasa C.	Memberikan hasil yang lebih baik dari metode grafik depedensi dan GREP, serta mendukung input pencarian multi teks.	Penelitian ini menggunakan kode & komentar untuk dijadikan *corpus* dan menggunakan LSI untuk mencari kedekatan konsep yang dicari. Hasilnya lebih baik dari metode grafik depedensi dan GREP.
Wilde dkk	Melakukan percobaan beberapa metode dalam mencari lokasi konsep & memberikan review kelebihan & kekurangan masing-masing.	Hasilnya dapat memberikan gambaran kelebihan & kekurangan beberapa metode tersebut.	Penelitian ini memberikan review tentang kelebihan masing-masing.
Savage dkk	Meneliti mencari lokasi konsep pada kode Java menggunakan LDA.	Hasil berupa alat bantu yaitu Topicxp, mempermudah pemrogram dalam merawat perangkat lunak.	Mencari lokasi konsep pada kode Java menggunakan LDA, membuat alat bantu Topicxp yang memudahkan

			pemrogram dalam merawat perangkat lunak

Terdapat berbagai macam penelitian tentang konsep lokasi selain yang disebutkan diatas, namun penelitian-penelitian tersebut diatas adalah yang relevan dengan penelitian ini.

2.3 Corpus Bahasa Alami

Corpus menurut kamus OxfordDictionaries adalah kumpulan dari teks (Oxford, 2014). *Corpus* banyak digunakan dalam objek penelitian terutama dibidang temu kembali informasi. Lin dkk membuat repositori *corpus* yang berasal dari ektraksi Javadoc. Isi dari dokumen Javadoc dipecah-pecah menjadi *corpus.* Kemudian pemrogram memasukkan kata atau kalimat yang akan dicari. Setelah itu dengan menggunakan algoritma LSI (*Latent Semantic Indexing*) dilakukan pencocokan kalimat dengan *corpus.* Setelah itu didapatkan *corpus* yang paling dekat dengan kalimat pencarian tersebut. Penelitian tersebut adalah penelitian yang akan diadopsi pada penelitian ini untuk mendapatkan *corpus* pada pertanyaan dan jawaban.

2.4 Corpus Potongan Kode

Corpus yang tidak kalah penting adalah *corpus* potongan kode. Potongan kode dipecah-pecah berdasarkan nama variabel, metode, kelas serta komentar. Berikut ini adalah penelitian terdahulu yang diadopsi dalam penelitian ini dalam rangka mendapatkan *corpus* potongan kode. Subramanian dkk (Subramanian, 2013) telah melakukan penelitian untuk membuat potongan kode pada StackOverflow menjadi masuk akal. Mereka menggunakan pengembangan kerangka kerja PPA(*Partial Program Analysis*) yaitu bagaimana mendapatkan variabel, metode, maupun kelas dari sepotong kode sumber. Mereka menggunakan beberapa alat bantu untuk membuat *Abstract Syntact Trees* yaitu pohon sintak yang berisi metode, variabel dari sepotong kode. Metode ini yang akan digunakan dalam penelitian ini untuk membuat *corpus* potongan kode.

Untuk mengatasi permasalahan penamaan metode Bogdan dkk (Bogdan Dit, 2011) telah meneliti bagaimana pengaruh pemotongan nama variabel, metode dapat membantu dalam menemukan konsep lokasi pada kode Mereka menguji beberapa metode seperti *CamelCase, Samurai, & TIDIER (Term Identifier Recognizer)* dengan beberapa dataset untuk melihat seberapa presisi metode tersebut dapat menjelaskan konsep lokasi. Penelitian ini menggunakan metode *CamelCase* untuk memotong nama variabel pada potongan kode yang ada pada StackOverflow.

Komentar dalam potongan kode juga termasuk dalam corpus potongan kode. Untuk mendapatkan komentar dalam potongan kode dapat dilakukan dengan menggunakan pendekatan yang dilakukan oleh Steidl dkk (Steidl, 2013). Mereka menggunakan pencocokan teks dengan metode membuat pola dengan Regular Expression yaitu:

- untuk mencari komentar dalam metode menggunakan pola [a-zA-Z]+\\.[a-zA-Z] +\\(.*\\),
- untuk mencari komentar dalam *if* atau *while* menggunakan pola (if\\s*\\(.*)| (while\\s*\\(.*), dan
- juga untuk yang berakhiran ; atau { atau mengandung ‘=, ==, ;, atau void.

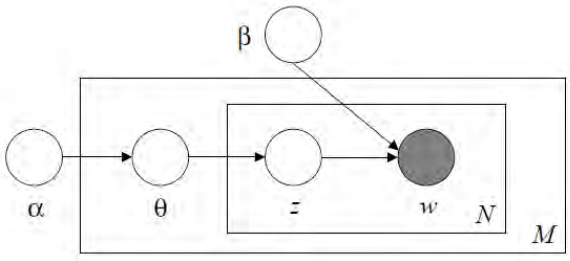
2.5 Latent Dirichlet Allocation

LDA (*Latent Dirichlet Allocation)* adalah model probabilistik generatif untuk sekelompok data diskrit seperti *corpus* (David M. Blei, 2003). LDA digunakan untuk mencari topik laten dalam *corpus.* Dalam LDA, sebuah dokumen terdiri dari kumpulan topik yang acak. Dokumen juga merupakan kumpulan kata yang merupakan gambaran dari topik-topik yang bersifat laten. Dalam membuat model generatif, LDA memberi asumsi bahwa masing-masing dokumen harus melalui proses sebagai berikut.

1) Gambar distribusi topik menggunakan distribusi Dirichlet dengan parameter α, $\theta_d \sim \text{Dir}(\alpha)$. Untuk perhitungan distribusi Dirichlet dapat melihat tutorial dari Bela dkk (Bela A. Frigyik, 2010).

2) Untuk masing-masing kata dalam dokumen dilakukan proses berikut:
   a) pilih sebuah topik z_n ~ Multinomial(θ), menggunakan distribusi multinomial,
   b) pilih sebuah kata w_n dari p(wn |zn,β), yaitu probabilitas multinomial yang dikondisikan pada topik z_n.

   Gambar berikut merupakan representasi grafik dari LDA.



Gambar 2. 1 Model Reprentasi Grafis pada LDA (Blei,2003).

Berikut ini adalah penjelasan gambar 2.1. Kotak tersebut adalah pelat yang mewakili proses perulangan. Pelat luar merupakan perulangan pada dokumen, sedangkan pelat dalam merupakan proses memilih topik-topik dan kata-kata yang berulang-ulang dalam dokumen. Sehingga terdapat perulangan bersarang untuk mendapatkan topik dalam kata dan dokumen. Variabel α dan β adalah parameter level *corpus* yang digunakan untuk perhitungan distribusi Dirichlet, diasumsikan untuk menjadi sampel sekali saat membuat *corpus.* Variabel θ adalah proporsi distribusi topik pada dokumen ke i . Variabel z adalah topik dari kata ke-j dari dokumen ke i. Variabel w adalah kata tersebut. N adalah dokumen ke-i. M adalah kumpulan dokumen. Berikut ini adalah persamaan-persamaan dalam LDA.

$$p(\theta|\alpha) = \frac{\Gamma(\sum_{i=l}^{k} \alpha_i)}{\prod_{i=l}^{k} \Gamma(\alpha_i)} \theta_l^{\alpha_l - 1} \dots \theta_k^{\alpha_k - l}, \qquad (2,1)$$

dimana persamaan tersebut adalah menghitung probabilitas proporsi topik dalam dokumen dengan parameter tertentu α . θ adalah proporsi topik dalam dokumen ke-i. α adalah proporsi parameter distribusi topik. α menentukan rasio tingkat kedekatan dengan topik tertentu, jika ada 3 topik dengan α masing-masing 1/3 maka distribusi akan berada di tengah-tengah *simplex*. *k* adalah jumlah topik yang telah ditetapkan. l adalah penugasan topik pada tiap kata.

$$p(\theta, z, w|\alpha, \beta\ ) = p(\theta|\alpha) \prod_{n-1}^{N} p(z_n|\theta) p(w_n|Z_n, \beta), \qquad (2,2)$$

Persamaan diatas menunjukkan perhitungan untuk mencari peluang distribusi gabungan topik pada kata dalam dokumen dengan parameter α, β yaitu distribusi proporsi dan jumlah topik. Z merupakan penugasan topik kedalam kata W. N adalah sejumlah kata W dalam sebuah dokumen.

$$p(w|\alpha, \beta\ ) = \int p(\theta|\alpha) \left(\prod_{n-1}^{N} \sum_{Z_n} p(z_n|\theta) p(w_n|Z_n, \beta) \right) d\theta, \qquad (2,3)$$

Persamaan diatas menunjukkan perhitungan untuk mencari peluang distribusi penugasan topik pada kata W dengan parameter α, β yaitu distribusi proporsi dan jumlah topik pada dokumen.

$$p(D|\alpha, \beta\ ) = \prod_{d=1}^{M} \int p(\theta_d|\alpha) \left(\prod_{n-1}^{N} \sum_{Z_n} p(z_{dn}|\theta_d) p(w_{dn}|Z_{dn}, \beta) \right) d\theta_d, \qquad (2,4)$$

Persamaan diatas merupakan perhitungan untuk mencari peluang dalam corpus dengan parameter α, β yaitu distribusi proporsi dan jumlah topik pada dokumen. Berikut ini adalah cara kerja LDA dalam menentukan topik secara garis besar.

1. Kita harus menentukan kira-kira berapa topik yang ada dalam dokumen (biasanya 100 tergantung jumlah dokumen).
2. Dalam setiap dokumen kemudian dibuat kelompok sebanyak 100 untuk kemudian masing-masing kata diberi salah satu topik acak dari 100 topik. Setelah itu dihitung peluang kemunculan topik dalam kata. Kemudian juga dihitung kemunculan topik dalam dokumen. Setelah itu dari kata-kata tersebut diganti topiknya dengan peluang topik yang baru sebesar mengalikan kedua peluang tersebut. Sehingga semakin banyak iterasi maka semakin kecil kemungkinan pergantian topik dari kata.
3. Proses ini dilakukan berulang sampai n kali. Setelah selesai kemudian dokumen tersebut dihitung berapa prosentase kata-kata yang masuk topik 1,2 dst. Sehingga didapatkan hasil misalnya dokumen i terdiri dari topik 1 20 %, topik 2 50 % dan 3 30%.

2.5.1 *Mallet Topic Modelling*

Mallet Topic Modelling adalah alat bantu berbasis Java untuk mengimplementasikan LDA menggunakan teknik sampling Gibbs untuk mengekstrak topik dalam *corpus* (McCallum, 2002) dan telah digunakan oleh beberapa penelitian. Berikut ini adalah penelitian-penelitian tersebut.

Tabel 2. 6 Beberapa Penelitian Terdahulu yang Berkaitan dengan StackOverflow, Lda & Mallet.

Nama, tahun	Deskripsi	Hasil	Kesimpulan
Allamanis dkk tahun 2013	Penelitian ini menggunakan Mallet untuk mengetahui variasi pertanyaaan dalam StackOverflow secara empiris	Menghasilkan hubungan antara teks & topik-topik kode dengan jenis pertanyaan.	Gambaran variasi pertanyaan dalam StackOverflow & kaitannya dengan teks & topik dalam kode.
Nama, tahun	Deskripsi	Hasil	Kesimpulan
Thomas, dkk pada tahun 2012	Mengeksplorasi evolusi kode menggunakan topik model pada kode sumber dengan LDA dengan alat bantu Mallet.	Proses evolusi kode berkaitan erat dengan evolusi topik pada perangkat lunak.	Penelitian ini menemukan pemetaan antara evolusi kode dan evolusi topik.
Arun, dkk pada tahun 2010	Mencari angka alami jumlah topik dalam LDA.	Menemukan jumlah topik yang paling tepat dalam data bertipe text dan gambar	Penelitian ini merekomendasikan jumlah topik yang paling tepat untuk data bertipe text dengan jumlah dimensi tertentu (yaitu 20 topik

			untuk ±1525 dimensi).

Dari beberapa penelitian tersebut diatas maka dapat disimpulkan bahwa Mallet dapat digunakan untuk mendapatkan topik dengan algoritma LDA dalam *corpus* yang ada pada penelitian ini. Jumlah data dalam penelitian yaitu sekitar ±1600 dimana jumlahnya hampir sama dengan penelitian yang dilakukan Arun dkk (Arun, 2010) sehingga untuk mendapatkan hasil *precision & recall* yang terbaik maka jumlah topik yang dipilih adalah 20 topik.

2.6 Precision & Recall

Precision & recall adalah metode pengukuran tingkat efektifitas suatu sistem pencarian data (Christopher D. Manning, 2008). *Precision* adalah pengukuran kemampuan sistem dalam menampilkan data yang relevan dibanding data yang tampil. Berikut ini adalah persamaan untuk mendapatkan nilai *precision.*

$$Precision = \frac{Jumlah\ item\ yang\ tampil\ \&\ relevan}{Jumlah\ item\ yang\ tampil} \quad (2.5)$$

Recall adalah pengukuran kemampuan sistem dalam menampilkan bagian data yang relevan dibandingkan data yang benar-benar relevan. Berikut ini adalah persamaan untuk mendapatkan nilai *Recall.*

$$Recall = \frac{Jumlah\ item\ yang\ tampil\ \&\ relevan}{Jumlah\ item\ relevan} \quad (2.6)$$

Untuk mempermudah dalam mendapatkan *precision & recall* berikut ini adalah tabel kemungkinan pada sistem temu kembali informasi.

Tabel 2. 7 Tabel Kemungkinan dalam Sistem Temu Kembali Informasi.

	Relevan	Tidak relevan
Tampil	*True Positives (TP)*	*False Positives (FP)*
Tidak Tampil	*False Negatives (FN)*	*True Negatives (TN)*

Sehingga untuk mendapatkan nilai *Precision* dapat menggunakan persamaan berikut ini.

$$Precision = \frac{TP}{(TP + FP)} \quad (2.7)$$

Bentang nilai *precision* mulai dari nol sampai satu. Nol artinya sistem memiliki presisi rendah sedangkan satu artinya sistem mampu memberikan rekomendasi secara sempurna. Presisi rendah berarti banyak dokumen yang tampil namum tidak relevan. Untuk mendapatkan nilai *recall* dapat menggunakan persamaan berikut ini.

$$Recall = \frac{TP}{(TP + FN)} \tag{2.8}$$

Bentang nilai *recall* mulai dari mulai dari nol sampai satu. Nilai *recall* yang rendah berarti sistem melewatkan kesempatan pengguna menemukan data yang relevan, sebaliknya recall yang tinggi sistem banyak memberikan data sehingga besar kemungkinan mendapatkan nilai yang relevan. Sistem yang baik adalah sistem yang memberikan nilai *precision & recall* yang tinggi. Umumnya kondisi *precision* tinggi akan berdampak pada *recall* yang rendah dan sebaliknya.

2.7 Kesamaan *Cosine*

Kesamaan *Cosine* adalah metode untuk membandingkan kesamaan antara dua buah kalimat. Kesamaan ini menggunakan pemodelan berbasis ruang vektor (Christopher D. Manning, 2008). Kalimat yang dibandingkan dipetakan kedalam tabel vektor yaitu jumlah kemunculan kata pada masing kalimat. Berikut ini adalah persamaan untuk mendapatkan nilai kesamaan *Cosine.*

$$Cos(kal1, kal2) = \frac{\overrightarrow{kal1} + \overrightarrow{kal2}}{\left\|\overrightarrow{kal1}\right\|\left\|\overrightarrow{kal2}\right\|} \tag{2.9}$$

Dimana kal1 adalah kalimat pertama dan kal2 adalah kalimat kedua. Vektor kalimat didapatkan dengan membuat tabel kemunculan kata. Sebagai contoh kalimat satu "Saya sedang makan" dibandingkan dengan kalimat "Saya sedang tidur" maka tabelnya sebagai berikut.

Tabel 2.8 Tabel Kemunculan Kata.

	Saya	Sedang	Makan	Tidur
Kal1	1	1	1	0
Kal2	1	1	0	1

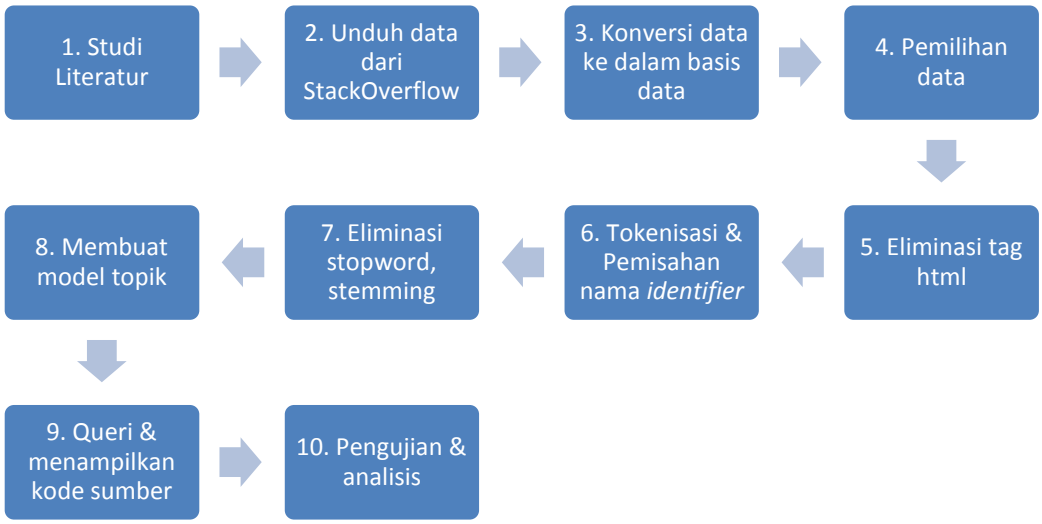
Dari tabel diatas maka kemiripan kalimat satu dengan dua adalah sebagai berikut.

$$Cos(kal1, kal2) = \frac{1 \times 1 + 1 \times 1 + 1 \times 0 + 0 \times 1}{\sqrt{1^2 + 1^2 + 1^2 + 0^2} \times \sqrt{1^2 + 1^2 + 0^2 + 1^2}} = \frac{2}{3} = 0{,}6$$

Sehingga tingkat kemiripan dari kalimat “Saya sedang makan” dibandingkan dengan kalimat “Saya sedang tidur” adalah 0,6. Kesamaan *Cosine* ini digunakan untuk membandingkan sebaran topik dari data dokumen dibandingkan dengan sebaran topik hasil inferensi data input dari pengguna

BAB 3
METODOLOGI PENELITIAN

3.1 Metodologi Penelitian



Gambar 3.1 Metodologi Penelitian

Gambar 3.1 menunjukkan metodologi penelitian yang akan dilakukan. Adapun uraian dari langkah diatas adalah sebagai berikut.

1. Studi literatur adalah proses mencari literatur tentang penelitian terdahulu. Penelitian yang dicari adalah penelitian yang berkaitan dengan pencarian lokasi konsep pada kode, StackOverflow, dan pemrosesan bahasa alami.
2. Unduh data dari stockoverflow adalah proses mengunduh data *dump* pertanyaan dan jawaban pada situs StackOverflow. Data ini tersedia pada situs https://archive.org/details/stackexchange. Data berupa file xml yang terdiri dari kolom dan baris.
3. Konversi data ke dalam basis data adalah proses dimana data tersebut kemudian dimasukkan ke dalam basis data MySql. Kemudian data dipilih khusus yang memiliki *tag Java* yang akan digunakan dalam penelitian ini. Dalam penelitian ini satu posting pertanyaan, jawaban dan kode merupakan

satu dokumen. Hal ini bertujuan untuk mendapatkan topik dalam satu posting tersebut.

4. Eliminasi *tag* html dilakukan untuk menghapus *tag* html dalam data. Hal ini dilakukan karena StackOverflow menuliskan *tag* dalam data. Eliminasi data dilakukan dengan pemrosesan *String* yaitu dengan membaca keseluruhan *string* dan jika menemui *tag* html maka *tag* dihapus.
5. Pemilihan data adalah proses menyeleksi data. Dalam struktur tabel hasil impor file xml terdapat *field tags* yaitu *field* yang berisi kategori dari *posting.* Data yang dipilih adalah data yang isi *field* mengandung kata Java agar data yang ada benar-benar merupakan khusus untuk bahasa pemrograman Java. Kemudian dengan bantuan Solr dipilih 159 data acak tentang *sort, database, file text, graphic & thread.* Solr mampu memberikan rekomendasi *posting* yang paling relevan berdasarkan algoritma tf-idf. Solr digunakan untuk mempercepat pencarian data acak.
6. Ekstraksi lokasi konsep dilakukan dengan mencari adanya kata yang terdiri dari lebih dari satu kata. Untuk mendeteksi *identifier* yang menggunakan aturan penamaan *camelCase* dapat dideteksi dengan operasi *string* yaitu dengan melihat apakah ada kemunculan huruf besar setelah huruf kecil dalam sebuah kata. Semisal ada kata *getName* maka akan menjadi *get name.* Sedangkan untuk pemisah menggunakan tanda _ maka dapat dilakukan dengan cara mencari apakah dalam kata terdapat tanda _, jika ada maka diganti dengan spasi. Semisal (*get_date*) maka akan menjadi *get date.*
7. Eliminasi Stopword adalah proses untuk menghilangkan kata yang sering muncul dalam data sebagai contoh *are, you, is, to, public, int*. Hal ini dilakukan dengan membaca setiap kata dalam data kemudian dibandingkan dengan daftar kata stopword. Jika kata tersebut terdapat dalam daftar maka kata tersebut dihapus, jika tidak maka dilanjutkan dengan kata selanjutnya. *Stemming* adalah mencari bentuk dasar dari kata, hal ini dilakukan agar variasi imbuhan kata dapat dieliminasi sehingga hanya terdapat bentuk kata dasarnya. Sebagai contoh kata *computing measurement* setelah dilakukan *Stemming* akan menjadi *comput measur*.

8. Membuat model topik dilakukan dengan algoritma LDA (*Latent Dirichlet Alocation)* dengan menggunakan Mallet. Daftar kata yang sudah mengalami eliminasi tag, ektraksi lokasi konsep, dan eliminasi *stopword* serta stemming akan menjadi model topik.
9. *Query* dan menampilkan kode sumber adalah proses untuk mencari kode sumber yang diinginkan. Selanjutnya *Query* diproses tokenisasi, eliminasi *stopword, stemming. Query* kemudian juga divalidasi apakah kata pada *Query* ada dalam pustaka model topik atau tidak. Proses validasi bertujuan untuk memfilter kata-kata yang tidak ada dalam pustaka model topik, sehingga jika *Query* diisi kata acak, maka sistem tidak merekomendasikan kode. Query kemudian diinferensi terhadap model topik yang telah ada sebelumnya. Hasilnya berupa sebaran proporsi topik dari kata tersebut, sebagai contoh kata "*map locate"* setelah dilakukan inferensi ternyata mengandung topik satu sebanyak 50 %, dan topik 2 sebanyak 30% dst. Hasil sebaran ini akan dicocokkan dengan kemiripan *cosine* dengan sebaran proporsi topik pada topik model, sehingga didapatkan ranking kemiripan antara input dengan dokumen-dokumen. Sistem akan menampilkan rangking kemiripan dokumen dengan ambang batas tertentu. Pengaturan ambang batas ini bertujuan agar dapat mengetahui seberapa jauh tingkat *precision* dan *recall* dari sistem ini.
10. Pengujian dan analisis dilakukan untuk menguji seberapa tingkat keberhasilan metode ini. Pengujian dilakukan dengan menggunakan pengukuran *precision & recall*.

3.2 Percobaan Awal

3.2.1 Studi Literatur

Proses pertama yaitu studi literatur dimana meliputi studi bagaimana mengunduh data dari StackOverflow, memproses kata dan membuat topik model menggunakan Mallet (implementasi LDA).

3.2.2 Unduh Data

Proses selanjutnya adalah mengunduh data pertanyaan, jawaban pada StackOverflow menggunakan API yang telah disediakan. Berikut ini adalah perintah untuk mendapatkan pertanyaan dengan no 1098383 yaitu dengan memasukkan format Url tertentu sesuai dengan contoh dari API "https://api.stackexchange.com/2.2/posts/109383?order=desc&sort=activity&site=StackOverflow&filter=withbody". Selanjutnya akan tampil data xml dari situs tersebut.


```
{
  "items": [
    {
      "owner": {
        "reputation": 1178,
        "user_id": 9466,
        "user_type": "registered",
        "profile_image":
"https://www.gravatar.com/avatar/725fa1e4f3e5fbac4cd60d24795fe15
a?s=128&d=identicon&r=PG",
        "display_name": "Abe",
        "link": "http://StackOverflow.com/users/9466/abe"
      },
      "score": 475,
      "last_edit_date": 1356706159,
      "last_activity_date": 1395250953,
      "creation_date": 1221944603,
      "post_type": "question",
      "post_id": 109383,
      "link": "http://StackOverflow.com/q/109383"
    }
  ]
```


Sedangkan jawaban dari pertanyaan tersebut didapatkan dengan API StackOverflow untuk mendapat jawaban 109383 dengan memasukkan alamat Url https://api.stackexchange.com/2.2/questions/109383/answers?order=desc&sort=votes&site=StackOverflow&filter=withbody


```
{
"items": [
{
"owner": {
"reputation": 1010,
"user_id": 157196,
"user_type": "registered",
"profile_image":
"https://www.gravatar.com/avatar/febc5371dd28abe7f1be33bfe7f0402
b?s=128&d=identicon&r=PG",
"display_name": "user157196",
"link": "http://StackOverflow.com/users/157196/user157196"
```

```
},
"is_accepted": false,
"community_owned_date": 1338338847,
"score": 235,
"last_activity_date": 1385225810,
"last_edit_date": 1385225810,
"creation_date": 1250409192,
"answer_id": 1283722,
"question_id": 109383,
"body": "<p><strong>Added note:</strong> <em>if you intend to use the code provided, be sure to read the comments as well to be aware of the implications.</em></p>\n\n<hr>\n\n<p>It seems much easier than all of the foregoing. Use a TreeMap as follows:</p>\n\n<pre><code>public class Testing {\n\n public static void main(String[] args) {\n\n HashMap&lt;String,Double&gt; map = new HashMap&lt;String,Double&gt;();\n ValueComparator bvc = new ValueComparator(map);\n TreeMap&lt;String,Double&gt; sorted_map = new TreeMap&lt;String,Double&gt;(bvc);\n\n map.put(\"A\",99.5);\n map.put(\"B\",67.4);\n map.put(\"C\",67.4);\n map.put(\"D\",67.3);\n\n System.out.println(\"unsorted map: \"+map);\n\n sorted_map.putAll(map);\n\n System.out.println(\"results: \"+sorted_map);\n }\n}\n\nclass ValueComparator implements Comparator&lt;String&gt; {\n\n Map&lt;String, Double&gt; base;\n public ValueComparator(Map&lt;String, Double&gt; base) {\n this.base = base;\n }\n\n // Note: this comparator imposes orderings that are inconsistent with equals. \n public int compare(String a, String b) {\n if (base.get(a) &gt;= base.get(b)) {\n return -1;\n } else {\n return 1;\n } // returning 0 would merge keys\n }\n}\n</code></pre>\n\n<p>Output:\n <pre>\n unsorted map: {D=67.3, A=99.5, B=67.4, C=67.4}\n results: {D=67.3, B=67.4, C=67.4, A=99.5}\n </pre></p>\n"
}
  "has_more": true,
    "quota_max": 300,
    "quota_remaining": 298
}
```


Dari jawaban pertama dan kedua dari StackOverflow tersebut kemudian dapat diambil data dokumen sebagai berikut.

```
“I am relatively new to Java, and often find that I need to sort a Map on the values. Since the values are not unique, I find myself converting the keySet into an array, and sorting that array through array sort with a custom comparator that sorts on the value associated with the key. Is there an easier way?
Added note: if you intend to use the code provided, be sure to read the comments as well to be aware of the implications. It seems much easier than all of the foregoing. Use a TreeMap as follows:
public class Testing {
    public static void main(String[] args) {
        HashMap<String,Double> map = new HashMap<String,Double>();
        ValueComparator bvc =  new ValueComparator(map);
        TreeMap<String,Double> sorted_map = new TreeMap<String,Double>(bvc);
```

```
        map.put("A",99.5);
        map.put("B",67.4);
        map.put("C",67.4);
        map.put("D",67.3);
        System.out.println("unsorted map: "+map);
        sorted_map.putAll(map);
        System.out.println("results: "+sorted_map);}}
class ValueComparator implements Comparator<String> {
    Map<String, Double> base;
    public ValueComparator(Map<String, Double> base) {
        this.base = base;
    }
    // Note: this comparator imposes orderings that are inconsistent
with equals.
    public int compare(String a, String b) {
        if (base.get(a) >= base.get(b)) {
            return -1;
        } else {
            return 1;
        } // returning 0 would merge keys
    }
}".
```

Berikut ini adalah perintah untuk mendapatkan pertanyaan dengan no 2784514 yaitu dengan memasukkan format Url tertentu sesuai dengan contoh https://api.stackexchange.com/2.2/posts/2784514?order=desc&sort=activity&site=StackOverflow&filter=withbody.


```
{
    "items": [
        {
            "owner": {
                "reputation": 2851,
                "user_id": 253387,
                "user_type": "registered",
                "accept_rate": 83,
                "profile_image":
"https://www.gravatar.com/avatar/a2ec41189711b656ebd6d60e26e1da07?
s=128&d=identicon&r=PG",
                "display_name": "Samuel",
                "link":
"http://StackOverflow.com/users/253387/samuel"
            },
            "score": 291,
            "last_edit_date": 1336009170,
            "last_activity_date": 1394723967,
            "creation_date": 1273180176,
            "post_type": "question",
            "post_id": 2784514,
            "link": "http://StackOverflow.com/q/2784514",
            "body": "<p>I read about sorting ArrayLists using a
Comparator but in all of the examples people used
```

```
<code>compareTo</code> which according to some research is a method for Strings.</p>\n\n<p>I wanted to sort an ArrayList of custom objects by one of their properties: a Date object\n(<code>getStartDay()</code>). Normally I compare them by <code>item1.getStartDate().before(item2.getStartDate())</code> so I was wondering whether I could write something like:</p>\n\n<pre><code>public class customComparator {\n public boolean compare(Object object1, Object object2) {\n return object1.getStartDate().before(object2.getStartDate());\n }\n}\n\npublic class randomName {\n ...\n Collections.sort(Database.arrayList, new customComparator);\n ...\n}\n</code></pre>\n\n<p>I just started with Java so please forgive my ignorance.</p>\n"
        }
    ],
    "has_more": false,
    "quota_max": 300,
    "quota_remaining": 298
}
```


Kemudian untuk jawabannya yaitu sebagai berikut.


```
{
    "items": [
        {
            "owner": {
                "reputation": 78203,
                "user_id": 13531,
                "user_type": "moderator",
                "accept_rate": 75,
                "profile_image":
"https://www.gravatar.com/avatar/49f3a2065659f5b4efa05ae29bd0856a?s=128&d=identicon&r=PG",
                "display_name": "Michael Myers",
                "link":
"http://StackOverflow.com/users/13531/michael-myers"
            },
            "is_accepted": true,
            "score": 413,
            "last_activity_date": 1363153281,
            "last_edit_date": 1363153281,
            "creation_date": 1273180703,
            "answer_id": 2784576,
            "question_id": 2784514,
            "body": "<p>Since <a
href=\"http://docs.oracle.com/javase/6/docs/api/java/util/Date.html\">Date</a> implements <a
href=\"http://docs.oracle.com/javase/6/docs/api/java/lang/Comparable.html\">Comparable</a>, it has a <code>compareTo</code> method just like <code>String</code> does.</p>\n\n<p>So your custom comparator could look like this:</p>\n\n<pre><code>public class CustomComparator implements Comparator&amp;lt;MyObject&amp;gt; {\n @Override\n public int compare(MyObject o1, MyObject o2) {\n return o1.getStartDate().compareTo(o2.getStartDate());\n }\n}\n</code></pre>\n\n<p>(The <code>compare()</code> method must return an <code>int</code>, so you couldn't directly return a
```

```
 <code>boolean</code>  like  you  were  planning  to
anyway.)</p>\n\n<p>Your sorting code would be just about like you
wrote:</p>\n\n<pre><code>Collections.sort(Database.arrayList, new
CustomComparator());\n</code></pre>\n\n<p> <br>\nA couple of
smaller  points  which  are  not  directly  related  to  the
question:</p>\n\n<ol>\n<li>By convention, classes start with an
upper-case letter while methods and variables start with a lowercase
letter.  That's why I changed the name of the comparator to
<code>CustomComparator</code>.</li>\n<li>Use  the  <a
href=\"http://docs.oracle.com/javase/6/docs/api/index.html\">Javad
ocs</a>. They will be invaluable if you keep working with
Java.</li>\n</ol>\n"  }
```

3.2.3 Konversi Data ke dalam Basis Data

Proses ini bertujuan untuk masukkan data yang telah diunduh kedalam database MySql. Proses ini meliputi pemilahan *tag* dalam file XML, selanjutnya informasi dalam *tag-tag* tersebut dipetakan kedalam *field-field* untuk kemudian disimpan pada tabel dalam database MySql. Tujuan penggunaan MySql pada proses ini akan memudahkan pemrosesan kata, pencarian data dan pengelompokan data.

3.2.4 Pemilihan Data

Pemilihan data adalah proses untuk memilih dokumen yang akan digunakan menjadi data uji dalam penelitian ini. Data dipilih dengan menggunakan bahasa Sql berdasarkan syarat yang ditentukan (menggunakan *select from where field='* syarat'*)*. Pada percobaan awal tidak dilakukan pemilihan data oleh karena hanya menggunakan dua dokumen yang diunduh dari StackOverflow. Sehingga data langsung saja digunakan untuk proses berikutnya.

3.2.5 Eliminasi Tag Html

Proses selanjutnya adalah eliminasi *tag* html. Proses ini dilakukan dengan menggunakan perintah SQL oleh karena data sudah tersimpan dalam tabel pada database MySql. Setelah dilakukan eliminasi, dokumen berubah menjadi sebagai berikut.

```
Sort ArrayList of custom Objects by property
I read about sorting ArrayLists using a Comparator but in all of
the examples people used compareTo which according to some research
is a method for Strings. I wanted to sort an ArrayList of custom
objects by one of their properties: a Date object (getStartDay()).
Normally I compare them by
```

```
item1.getStartDate().before(item2.getStartDate())    so    I    was
wondering whether I could write something like:
public class customComparator {
public boolean compare(Object object1, Object object2) {
    return object1.getStartDate().before(object2.getStartDate());}
}
public class randomName {
    ...
    Collections.sort(Database.arrayList, new customComparator);
}
I just started with Java so please forgive my ignorance.
Since Date implements Comparable, it has a compareTo method just
like String does.
So your custom comparator could look like this:
public class CustomComparator implements Comparator<MyObject> {
    @Override
    public int compare(MyObject o1, MyObject o2) {
        return     o1.getStartDate().compareTo(o2.getStartDate());
}}
(The compare() method must return an int, so you couldn't directly
return a boolean like you were planning to anyway.)
Your sorting code would be just about like you wrote:
Collections.sort(Database.arrayList, new CustomComparator());
A couple of smaller points which are not directly related to the
question:
    By convention, classes start with an upper-case letter while
methods and variables start with a lower-case letter. That's why I
changed the name of the comparator to CustomComparator.
    Use the Javadocs. They will be invaluable if you keep working
with Java.
```

3.2.6 Tokenisasi & Pemisahan Nama *Identifier*

Proses selanjutnya adalah tokenisasi dan pemisahan nama identifier pada dokumen pertanyaan dan jawaban diatas. Proses yang pertama adalah tokenisasi yaitu proses memilah-milah kalimat menjadi kata-kata/token tertentu. Proses memilah pertanyaan/jawaban untuk selain kode program dilakukan dengan cara biasa yaitu menghilangkan tanda baca seperti koma, titik, dll. Sedangkan untuk pemisahan nama *identifier* pada kode program dilakukan dengan memperhatikan aturan penamaan variabel seperti *camelCase*, penggunaan tanda _,- sebagai pemisah antara token satu dengan yang lain. Berikut ini daftar token dari hasil proses tokenisasi.

Tabel 3.1 Daftar Token Setelah Tokenisasi.

I	*I*	*associated*	*as*	*Testing*	*Map*
am	*find*	*with*	*well*	*public*	*String*
relatively	*myself*	*the*	*to*	*static*	*Double*

new	*converting*	*key*	*be*	*void*	*sorted*
to	*the*	*Is*	*aware*	*main*	*map*
Java	*keySet*	*there*	*of*	*String*	*new*
and	*into*	*an*	*the*	*args*	*Tree*
often	*an*	*easier*	*implications*	*Hash*	*Map*
find	*array*	*way*	*It*	*Map*	*String*
that	*and*	*Added*	*seems*	*String*	*Double*
I	*sorting*	*note*	*much*	*Double*	*bvc*
need	*that*	*if*	*easier*	*map*	*map*
to	*array*	*you*	*than*	*new*	*put*
sort	*through*	*intend*	*all*	*Hash*	*99.5*
a	*array*	*to*	*of*	*Map*	*map*
Map	*sort*	*use*	*the*	*String*	*put*
on	*with*	*the*	*foregoing*	*Double*	*67.4*
the	*a*	*code*	*Use*	*Value*	*map*
values	*custom*	*provided*	*a*	*Comparator*	*put*
Since	*comparator*	*be*	*Tree*	*bvc*	*67.3*
the	*that*	*sure*	*Map*	*new*	*map*
values	*sorts*	*to*	*as*	*Value*	*sorted*
are	*on*	*read*	*follows*	*Comparator*	*map*
not	*the*	*the*	*public*	*map*	*put*
unique	*value*	*comments*	*class*	*Tree*	*All*
map	*public*	*class*	*b*	*String*	*b*
System	*int*	*Value*	*if*	*Double*	*return*
out	*compare*	*Comparator*	*base*	*base*	*-1.0*
println	*String*	*implements*	*get*	*public*	*else*
sorted	*a*	*Comparator*	*a*	*Value*	*return*
map	*String*	*String*	*base*	*Map*	*base*
Map	*get*	*Comparator*	*1.0*	*String*	
Double	*base*	*this*			

3.2.7 Eliminasi *Stopword & Stemming*

Proses berikutnya adalah eliminasi *stopword* yaitu menghilangkan kata-kata yang bersifat umum. Kata-kata tersebut dihilangkan karena tidak memiliki makna yang berarti. Selain itu untuk angka juga dihilangkan karena angka merupakan isi dari variabel dan tidak dibutuhkan. Sehingga dari daftar kata diatas direduksi menjadi daftar kata berikut ini.

Tabel 3.2 Daftar Token Setelah Stopword

relatively	*Intend*	*Comparator*	*map*
Java	*code*	*Map*	*Value*

find	*provided*	*Tree*	*Comparator*
sort	*sure*	*Map*	*Comparator*
Map	*read*	*Sorted*	*Map*
Since	*comments*	*Map*	*base*
unique	*well*	*Tree*	*Value*
find	*aware*	*Map*	*Comparator*
converting	*implications*	*Bvc*	*Map*
Set	*easier*	*Map*	*base*
key	*foregoing*	*Put*	*this*
array	*Tree*	*Map*	*base*
sorting	*Map*	*put*	*compare*
array	*follows*	*map*	*a*
through	*Testing*	*put*	*b*
array	*main*	*map*	*base*
sort	*Hash*	*sorted*	*a*
Custom	*Map*	*map*	*base*
comparator	*map*	*put*	*B*
sorts	*Hash*	*All*	*couldn*
associated	*Map*	*map*	*code*
key	*Value*	*System*	*wrote*
easier	*Comparator*	*out*	*couple*
Added	*bvc*	*println*	*list*
Note	*Value*	*sorted*	*smaller*
start	*list*	*wanted*	*points*
date	*item*	*properties*	*directly*
custom	*before*	*day*	*related*
comparator	*could*	*normally*	*question*
object	*collections*	*wondering*	*convention*
get	*database*	*whether*	*classes*
compare	*java*	*write*	*while*
public	*implements*	*something*	*methods*
sort	*your*	*boolean*	*variables*
like	*letter*	*random*	*lower*
array	*case*	*name*	*use*
method	*property*	*started*	*javadocs*
class	*read*	*please*	*invaluable*
return	*using*	*forgive*	*keep*
just	*examples*	*ignorance*	*working*
object	*used*	*since*	*upper*
lists	*compareto*	*comparable*	*int*
objects	*according*	*string*	
about	*some*	*does*	

sorting	*research*	*look*
array	*strings*	*override*

Proses selanjutnya adalah *stemming*, yaitu proses untuk menghapus akhiran sehingga hanya tinggal kata dasar saja. Sehingga kata-kata menjadi kata-kata pada daftar berikut.

Tabel 3.3 Daftar Token Setelah *Stemming*.

relat	*Intend*	*compar*	*map*
java	*code*	*map*	*valu*
find	*provid*	*tree*	*compar*
sort	*sure*	*map*	*compar*
map	*read*	*sort*	*map*
sinc	*comment*	*map*	*base*
uniqu	*well*	*tree*	*valu*
find	*awar*	*map*	*compar*
convert	*implic*	*bvc*	*map*
set	*easier*	*map*	*base*
key	*forego*	*put*	*this*
array	*tree*	*map*	*base*
sort	*map*	*put*	*compar*
array	*follow*	*map*	*a*
through	*test*	*put*	*b*
array	*main*	*map*	*base*
sort	*hash*	*sort*	*a*
custom	*map*	*map*	*base*
compar	*map*	*put*	*b*
sort	*hash*	*all*	*couldn*
associ	*map*	*map*	*code*
key	*valu*	*system*	*wrote*
easier	*compar*	*out*	*coupl*
ad	*bvc*	*println*	*list*
note	*valu*	*sort*	*smaller*
start	*list*	*want*	*points*
date	*item*	*properti*	*direct*
custom	*befor*	*day*	*relat*
compar	*could*	*normal*	*question*
object	*collect*	*wonder*	*convent*
get	*databas*	*whether*	*classes*
compar	*java*	*write*	*while*
public	*implement*	*someth*	*method*
sort	*your*	*boolean*	*variabl*

like	*letter*	*random*	*lower*
array	*case*	*name*	*use*
method	*properti*	*start*	*javadocs*
class	*read*	*pleas*	*invalu*
return	*use*	*forgiv*	*keep*
just	*exampl*	*ignor*	
object	*use*	*sinc*	
lists	*compare*	*compar*	
objects	*accord*	*string*	
about	*some*	*does*	
sort	*research*	*look*	
array	*string*	*overrid*	
work	*upper*	*int*	

3.2.8 Membuat Model Topik

Setelah itu adalah proses pembuatan model topik pada dokumen menggunakan algoritma LDA dengan alat bantu Mallet. Berikut ini dokumen yang akan diproses.

Data1.txt

```
relat java sort map sinc uniq find convert set key array sort array through array sort custom compar sort associ key easi Ad note intend code provid sure read comment well to aware implic easi forego Tree Map follow test main hash map map hash map valu compar bvc valu compar map tree map sort map tree map bvc map put map put map put sort map map sort map put all map system out println sort map valu compar compar map base valu compar map base this base  compar a b base a base b
```

Data2.txt

```
start date  custom compar object get compar public sort like array method class return just object lists bjects about   sort   array work list item befor could collect databas java implement your letter case properti read use exampl use compare accord some research string upper want properti day normal wonder whether write someth boolean random name start pleas forgiv ignor sinc compar string does look overrid int couldn code wrote coupl list smaller points direct relat question convent classes while method variabl lower use javadocs invalu keep
```

Dengan jumlah topik dua maka pemetaan sebaran topik dijelaskan pada tabel 3.4 Pemetaan tersebut kemudian disimpan dalam file text.

Tabel 3.4 Daftar Pemetaan Model Topik & Sebaran Dokumen.

#doc name topic proportion ...		
1	file:/D:/mallet/data/text.txt	topik 1 0.59% topik 2 0.41 %
2	file:/D:/mallet/data/text2.txt	topik 1 0.52% topik 2 0.47 %

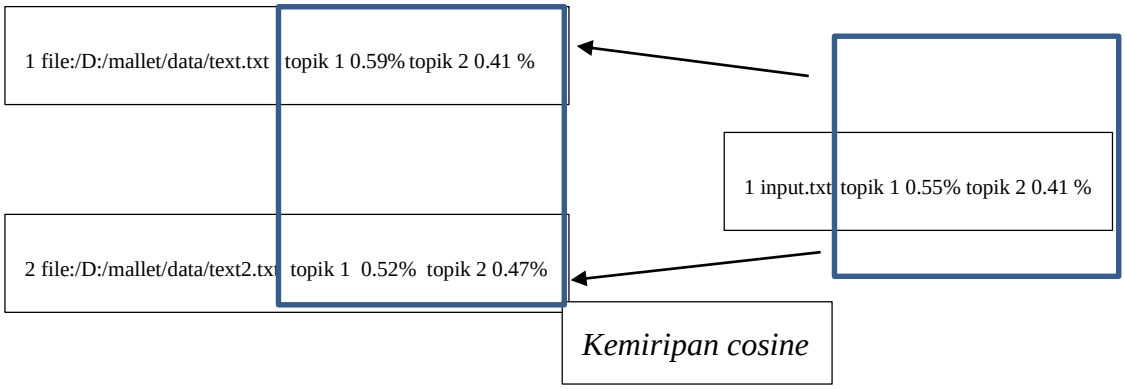
3.2.9 *Query* dan Menampilkan Kode Program

Pencarian oleh pengguna dilakukan dengan inferensi topik berdasarkan topik model dokumen. Sebagai contoh kata yang dicari adalah *Map compar*, maka sistem akan melakukan inferensi topik berdasarkan model topic sebelumnya, sehingga menghasilkan proporsi topik. Berikut ini adalah proporsi topik dari kata *Map compar* hasil inferensi sistem.

Tabel 3.5 Daftar Proporsi Topik Kata *Map Compar*.

#doc name topic proportion ...		
1	input.txt	topik 1 0.55% topik 2 0.41 %

Kemudian hasil tersebut dibandingkan kemiripannya dengan proporsi topik dari model topik sebelumnya dengan menggunakan metode kesamaan *Cosine*. Hal ini bertujuan untuk dapat mengetahui rangking kemiripannya. Bagian yang menjadi perbandingan adalah topik dan nilai proporsinya saja. Berikut ini adalah ilustrasi untuk mencari kemiripan sebaran topik.



Gambar 3.2 Ilustrasi Mencari Kemiripan *Cosine* Antara Proporsi Topik Dokumen Dengan Proporsi Topik Input dari Pengguna.

3.3 Pengujian dan Analisis Hasil

Untuk mengetahui keberhasilan metode ini maka dibutuhkan sebuah pengujian. Pengujian dilakukan dengan menggunakan *precision & recall.* Hasil dari rekomendasi sistem dibandingkan dengan hasil pakar. Skenario yang digunakan pada pengujian adalah mencari kode dengan lima pertanyaan. Empat pertanyaan relevan dan satu pertanyaan yang tidak relevan. Selain itu ambang batas kemiripan juga dijadikan parameter pengujian. Ambang batas mulai dari 0.1 sampai 0.9. Jumlah topik berjumlah iterasi juga digunakan untuk parameter. Iterasi mulai dari 1000 kelipatan dua sehingga menjadi 2000, 4000, 8000. Dengan demikian terdapat kombinasi yang cukup banyak untuk dapat menghasilkan pengukuran yang valid. Dari hasil pengujian kemudian dicatat tingkat *precision & recall* dari masing-masing pengaturan parameter tersebut.

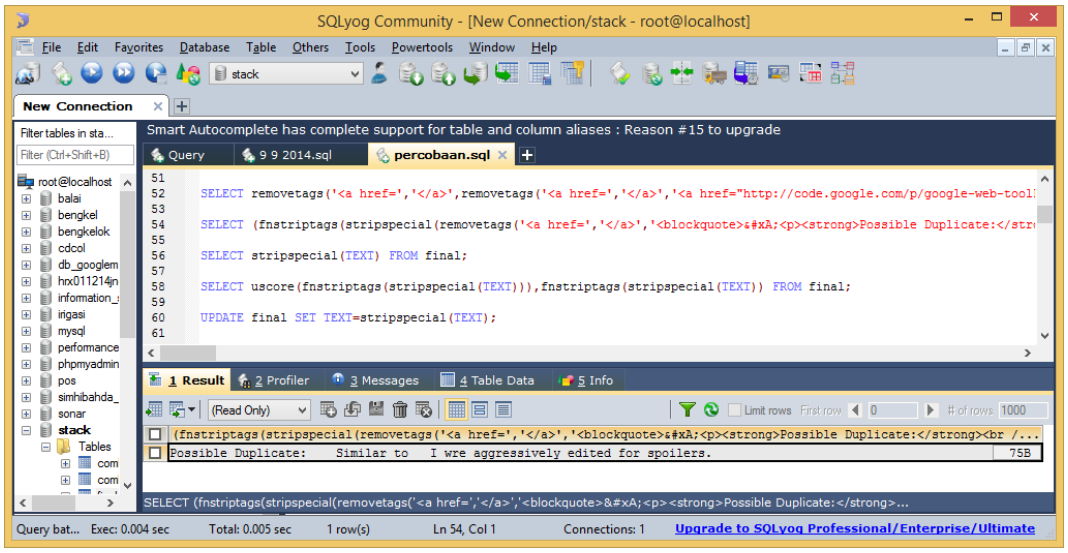
[Halaman ini sengaja dikosongkan]

BAB 4
HASIL PENELITIAN & PEMBAHASAN

4.1 Pengumpulan Dataset

Dataset yang digunakan adalah data *posting* permasalahan yang ada di media sosial StackOverflow. Data didapatkan dari backup yang disediakan StackOverflow pada tanggal 13-3-2014 bernama *file posts,7z* yaitu *file* terkompresi yang isinya semua *posting* di StackOverflow sampai dengan tanggal tersebut. Selanjutnya data tersebut dimasukkan ke dalam database MySql untuk dilakukan pra proses.

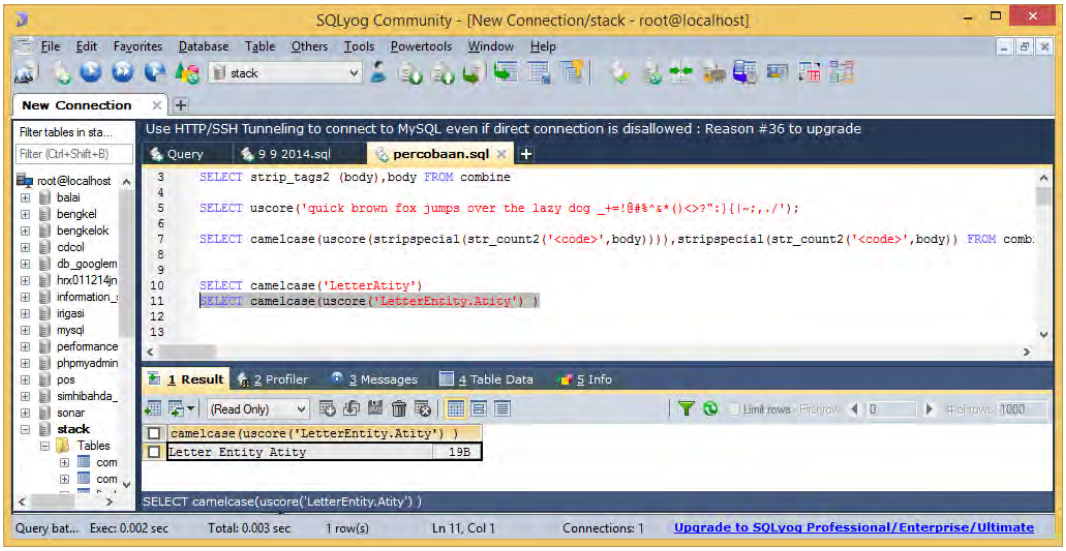
Pra proses dimulai dari pemilihan *posting* yang memiliki *tag* Java. Pemilihan dilakukan dengan menggunakan bahasa Sql. Kemudian karena data berisi text yang berformat html, maka dilakukan proses eliminasi *tag* html. Eliminasi *tag* html dilakukan dengan membuat *function removeTags, stripSpecial, fnstriptags* dalam MySql. Gambar 4.1 menunjukkan percobaan eliminasi tag html pada MySql.



Gambar 4. 1 Percobaan Fungsi Eliminasi *Tag* pada Fase Pra Proses.

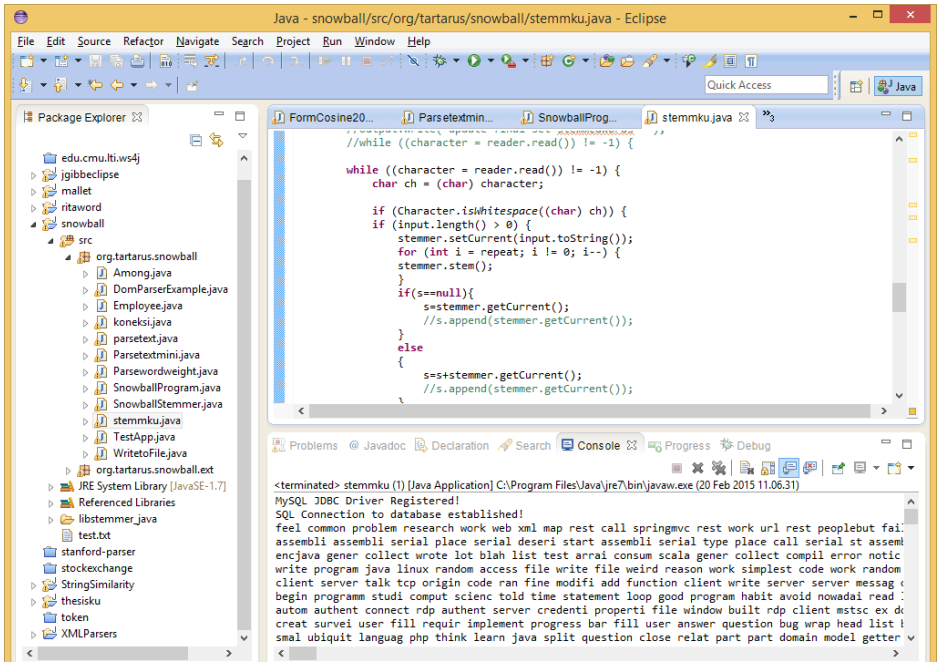
Pra proses selanjutnya adalah mencari lokasi konsep dalam isi *posting*. Lokasi konsep yang dicari adalah kata panjang yang kemungkinan terdiri dari kombinasi dua kata. Penamaan kata yang dicari adalah yang menggunakan penamaan *camelCase(contoh: getName)* & kata yang dipisah dengan tanda _

(contoh: *get_name).* Proses ini dilakukan dengan membuat *function camelCase & uscore (*isi dalam lampiran) dalam MySql. Gambar 4.2 menunjukkan percobaan pemisahan nama variabel yang menggunakan standar camelCase dan tanda (. _ -).



Gambar 4. 2 Percobaan Fungsi Pemisahan Nama Identifier pada Fase pra Proses.

Berikutnya adalah proses eliminasi *stopword*, *stemming.* Eliminasi *stopword* dilakukan dengan menggunakan queri sql yang disimpan dalam *function stopwords*. *Function stopwords* mengeliminasi kata-kata yang umum termasuk *reserved word* dalam bahasa pemrograman Java. Proses ini bertujuan agar kata-kata yang umum seperti *i, you, ask,* dst serta *reserved word* seperti *public, void, int* tidak ikut diproses. Selanjutnya adalah proses *Stemming* dimana kata-kata yang mengandung variasi imbuhan diambil kata dasarnya saja. *Stemming* dilakukan dengan menggunakan bantuan pustaka Snowball. Gambar 4.3 menunjukkan percobaan proses *stemming.* Setelah pemrosesan bahasa alami kemudian data selanjutnya dipilih lagi secara acak dengan kata kunci "*sort, database, file text, graphic, thread".* Pemilihan kata kunci tersebut dipilih oleh karena topik tersebut relatif mudah untuk pakar. Hal ini dilakukan oleh karena metode *precision & recall* memerlukan kepastian relevansi isi dokumen.



Gambar 4. 3 Percobaan Fungsi Stemming pada Fase Pra Proses.

Berikut ini adalah komposisi dataset dalam penelitian ini.

Tabel 4. 1 Komposisi Topik dalam Dataset.

No	Kata kunci topik	Jumlah
1	Sort	34
2	Database	34
3	File Text	32
4	Graphic	34
5	Thread	19
	Total *Posting*	**153**

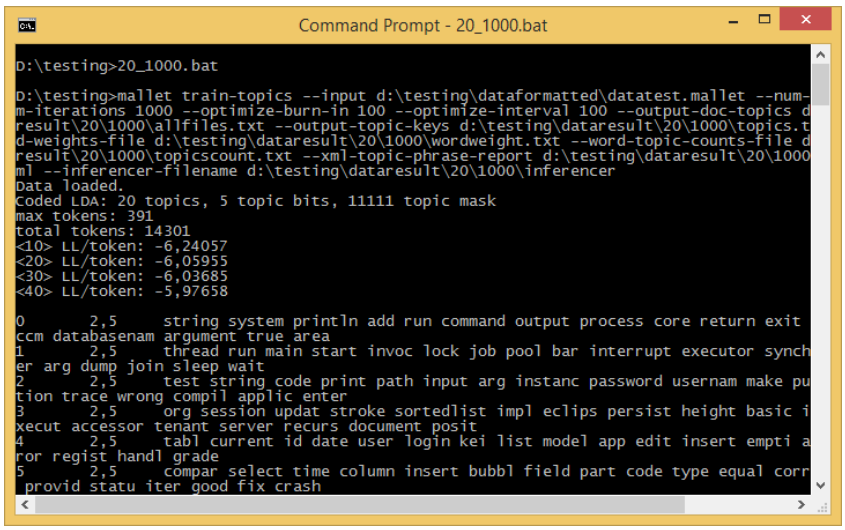
Secara kuantitas dataset terdiri dari ±14000 jumlah kata dengan dimensi kata (kata yang unik) sejumlah ±1600 dimensi kata.

Pakar kemudian membaca *posting* satu per satu dan membuat tinjauan dari masing-masing *posting*. Tinjauan tersebut kemudian dijadikan dasar dalam menentukan jawaban yang relevan pada fase pengujian sistem.

4.2 Model Topik

Model topik merupakan data yang dijadikan dasar dalam menentukan proporsi topik dari pencarian kata-kata dalam sistem ini. Dalam membuat model topik menggunakan LDA, menentukan jumlah topik merupakan faktor penting. Oleh sebab itu menentukan jumlah topik sebaiknya ditentukan dengan melihat jumlah dimensi kata. Arun, dkk menemukan bahwa dengan data berupa kata yang terdiri dari ±1500 dimensi maka jumlah topik yang cocok adalah 20 topik (Arun, 2010). Dalam penelitian ini menggunakan jumlah topik 20 karena jumlah dimensi mendekati jumlah dimensi dari data dalam penelitian mereka. Faktor lainnya yaitu jumlah iterasi dalam membuat model topik. Dari beberapa referensi belum dapat diketahui secara pasti berapa angka yang tepat dalam menentukan jumlah iterasi. Namun untuk mendapatkan hasil yang komprehensif, pada penelitian ini dipilih berbagai variasi iterasi. Variasi iterasi tersebut terdiri dari 1000,2000,4000,8000.

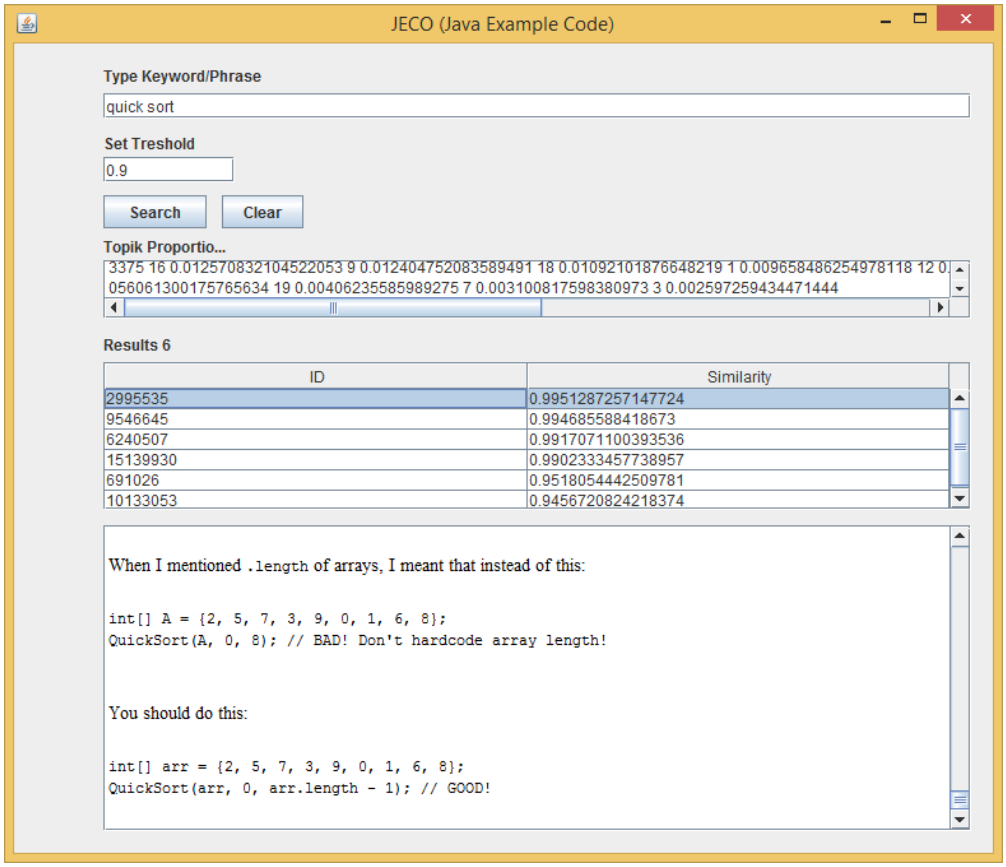
Untuk membuat model topik, data pada database di ekspor ke dalam bentuk file text dengan menggunakan perintah Sql. Kemudian model topik dari data dibuat dengan menggunakan Mallet dengan parameter *file* text tersebut diatas. Setelah itu hasilnya adalah *file* sebaran proporsi topik pada masing-masing *posting*. Hasil lainnya yaitu *inferencer* yang berisi susunan index kata-kata secara alfabetis dari topik model yang telah dibuat. Gambar 4.4 menunjukkan percobaan membuat model topik.



Gambar 4. 4 Percobaan Membuat Model Topik.

4.3 Pengembangan Aplikasi JECO (*Java Example Code)*

Dalam rangka memberikan kemudahan dalam menguji metode yang diajukan dalam thesis ini, maka dibuatlah JECO yaitu aplikasi desktop untuk menampilkan rekomendasi kode. JECO dikembangkan dengan bahasa Java SE, dengan menggunakan bantuan pustaka-pustaka seperti Snowball dan MALLET. Pengguna cukup memasukkan kata atau frase kedalam input kemudian JECO akan melakukan inferensi dengan model topik yang ada dan hasil inferensi akan ditampilkan kedalam *Grid* diurut berdasarkan tingkat kemiripan proporsi topik antara hasil inferensi dengan *posting*. Pada JECO juga terdapat fitur atur ambang batas kemiripan untuk mengeliminasi posting yang kurang dari ambang batas tersebut. Semisal ambang batas 0 artinya semua *posting* yang nilainya lebih dari 0 akan ditampilkan semua, sebaliknya jika 0,8 maka yang nilainya kurang dari 0,8 tidak akan tampil. Gambar 4.5 menunjukkan tampilan dari aplikasi JECO.



Gambar 4. 5 Penampilan Aplikasi JECO.

4.4 Pengujian

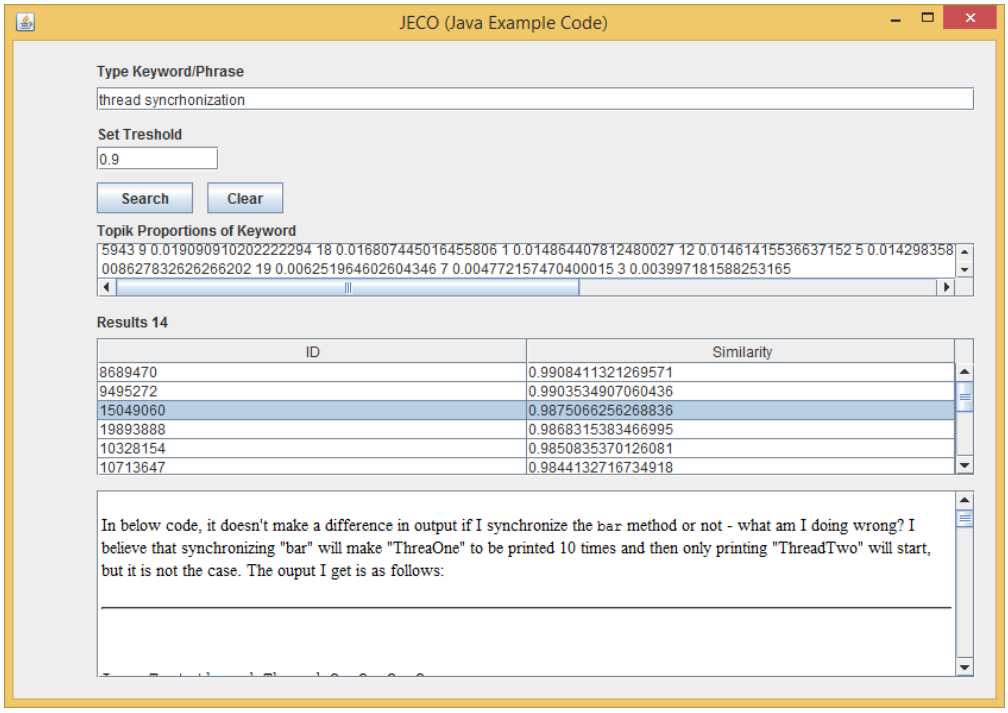
Untuk dapat mengukur tingkat *precision & recall* dari metode ini, maka dalam thesis ini dilakukan serangkaian percobaan. Percobaan yang dimaksud yaitu dengan menguji dengan lima pertanyaan pada masing-masing set model topik (iterasi 1000, iterasi 2000, iterasi 4000, iterasi 8000). Selain itu pada masing-masing iterasi juga diuji dengan ambang batas yang bervariasi (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9). Kelima pertanyaan tersebut adalah sebagai berikut.

1. *Thread synchronization.*
2. *Open connection on mysql.*
3. *Bubble sort.*
4. *Draw rectangle.*
5. *Music playlist.*

Dasar pemilihan pertanyaan satu sampai empat karena pertanyaan-pertanyaan tersebut merupakan pertanyaan umum serta relevan dengan isi *posting.* Namun sebaliknya untuk pertanyaan kelima sama sekali tidak relevan dengan isi *posting,* hal ini bertujuan untuk mengetahui bagaimana JECO mampu menjawab hal yang sama sekali tidak ada pada dataset.

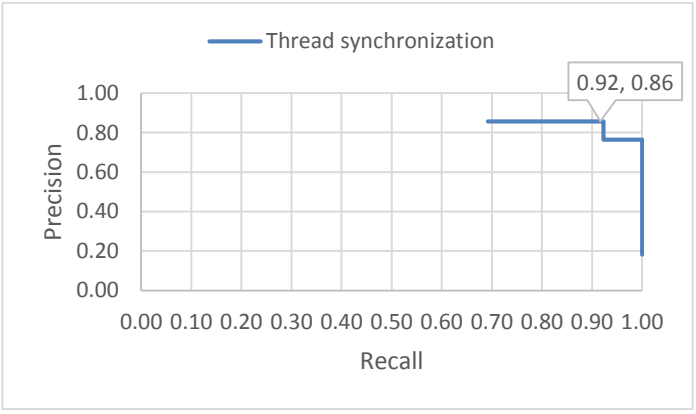
4.4.1 Pengujian dengan Pertanyaan *Thread Synchronization*

Pengujian pertama yaitu dengan pertanyaan *thread synchronization.* Maksud pertanyaan disini yaitu pengguna ingin mengetahui contoh kode dari implementasi *thread synchronization.* Pengujian dilakukan terhadap model topik dengan iterasi 1000, 2000, 4000, 8000. Selain itu pada masing-masing iterasi juga diuji dengan bermacam-macam ambang batas (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9). Tujuan dari masing-masing pengujian tersebut adalah untuk mendapatkan seberapa baik tingkat *precision & recall* yang dapat diraih dengan metode pada penelitian ini. Gambar 4.6 menunjukkan salah satu percobaan dengan pertanyaan *thread synchronization.*



Gambar 4. 6 Percobaan dengan Pertanyaan *Thread Synchronization*.

Hasil percobaan dengan pertanyaan *thread synchronization* ditunjukkan pada Gambar 4.7. yaitu plot kurva interpolasi *precision & recall*.

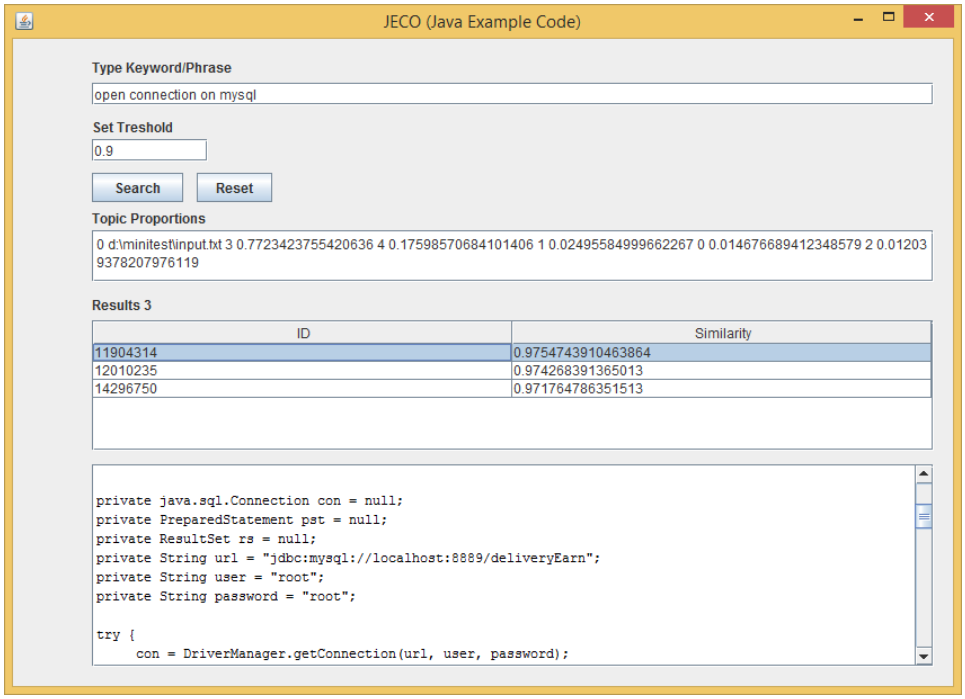


Gambar 4. 7 Plot Kurva Interpolasi *Precision & Recall* Hasil Percobaan dengan Pertanyaan *Thread Synchronization*.

Hasil percobaan pada pertanyaan pertama menunjukkan hasil yang paling baik dimana nilai *precision* terbaik adalah 0,86 dengan *recall* 0,92. Hasil ini didapatkan pada iterasi 2000 dan ambang batas 0,8. Hal ini terjadi karena 13 dari 19 *posting* dengan topik *thread* adalah tentang *thread synchronization.*

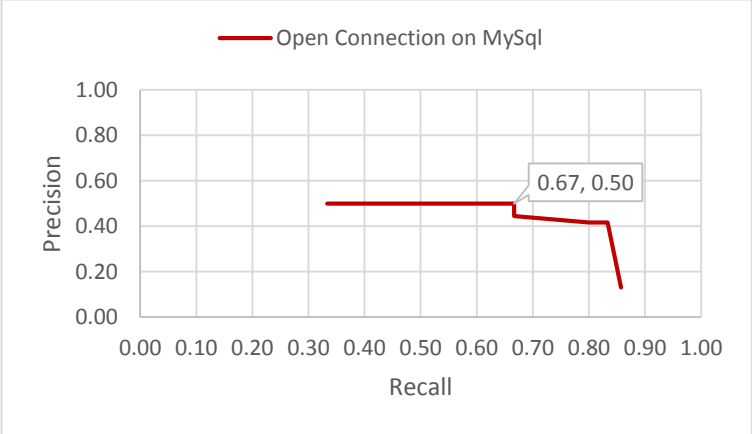
4.4.2 Pengujian dengan Pertanyaan *Open Connection on Mysql*

Pengujian selanjutnya adalah dengan menggunakan pertanyaan *open connection on mysql.* Maksud dari pertanyaan tersebut adalah ingin mengetahui contoh kode untuk implementasi membuat koneksi dengan database MySql pada Java. Pengujian dilakukan pada model topik yang bervariasi (1000, 2000, 4000, 8000) dan pada tiap iterasi dilakukan pengaturan ambang batas (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9). Gambar 4.8 menunjukkan salah satu percobaan menggunakan pertanyaan *open connection on mysql.*



Gambar 4. 8 Percobaan dengan Pertanyaan *Open Connection on MySql*.

Hasil percobaan dengan pertanyaan *open connection on mysql* ditunjukkan pada Gambar 4.9. yaitu plot kurva interpolasi *precision & recall* untuk pertanyaan *open connection on mysql*.

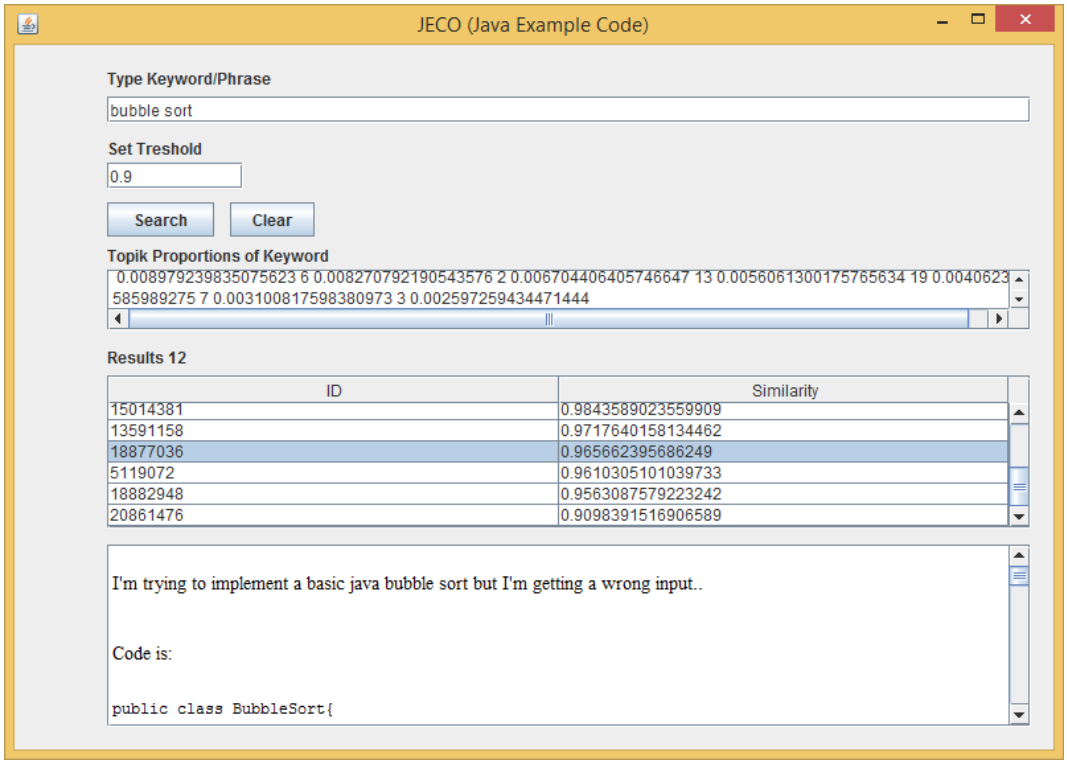


Gambar 4. 9 Plot Kurva Interpolasi *Precision & Recall* Hasil Percobaan dengan Pertanyaan *Open Connection on MySql*.

Hasil percobaan pada pertanyaan kedua menunjukkan hasil yang cukup dimana nilai *precision* terbaik adalah 0,50 dengan *recall* 0,67. Hasil ini didapatkan pada iterasi 4000 dan dengan ambang batas 0,3. Hal ini terjadi karena hanya 6 dari 34 *posting* dengan topik *database* adalah tentang *open connection on mySql.*

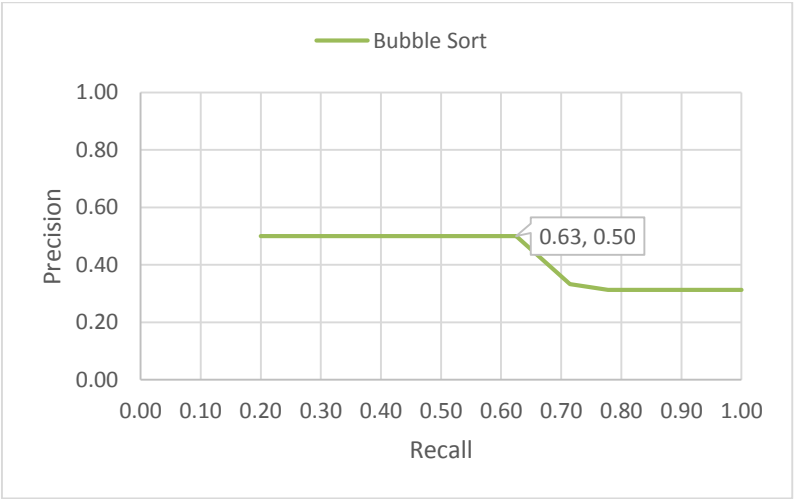
4.4.3 Pengujian dengan Pertanyaan *Bubble Sort*

Pengujian berikutnya adalah dengan pertanyaan *bubble sort.* Maksud dari pertanyaan tersebut adalah untuk mengetahui contoh kode implementasi *bubble sort* pada Java. Sama dengan pengujian-pengujian sebelumnya, pengujian ini juga menggunakan model topik dengan iterasi yang bervariasi(1000, 2000, 4000, 8000) dan pada tiap iterasi dilakukan pengaturan ambang batas (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9). Gambar 4.10 menunjukkan salah satu percobaan menggunakan pertanyaan *bubble sort.*



Gambar 4. 10 Percobaan dengan Pertanyaan *Bubble Sort*.

Setelah dilakukan percobaan pada model topik pada semua iterasi dan semua level ambang batas, maka didapatkan hasil *precision & recall* yang digambarkan pada gambar 4.11 yaitu plot kurva interpolasi *precision & recall* untuk pertanyaan *bubble sort.*

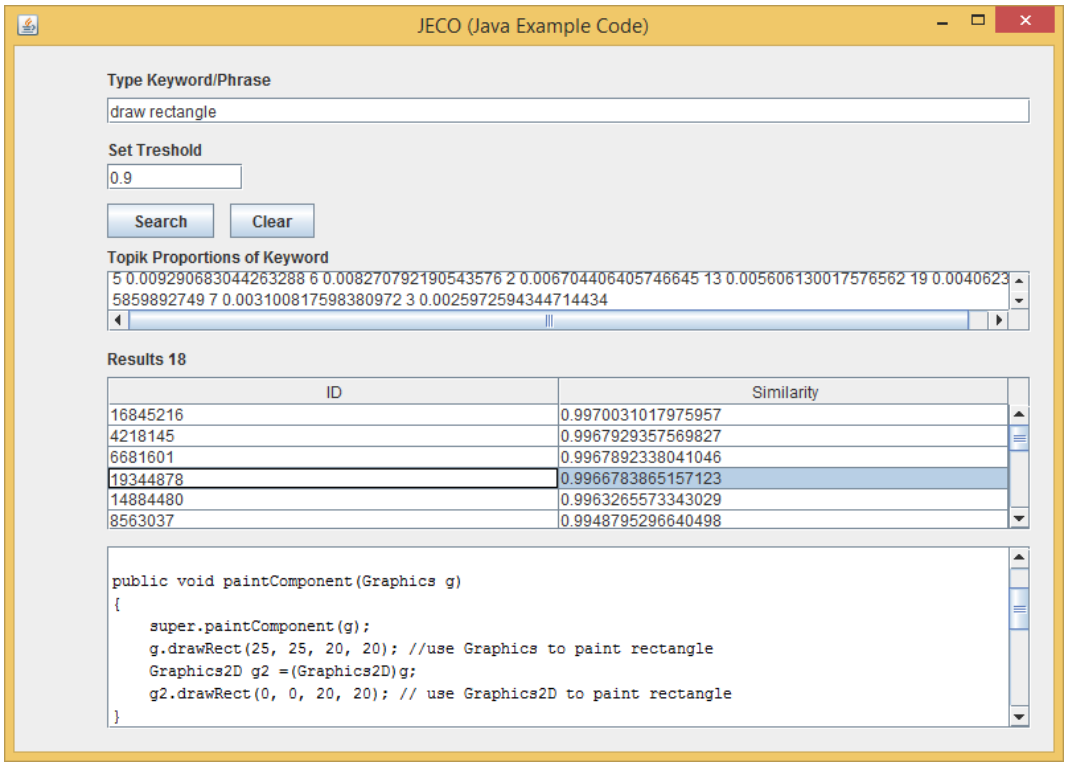


Gambar 4. 11 Plot Kurva Interpolasi Precision & Recall Hasil Percobaan dengan Pertanyaan *Bubble Sort*.

Hasil percobaan pada pertanyaan *bubble sort* menunjukkan hasil yang cukup dimana nilai *precision* terbaik adalah 0,50 dengan *recall* 0,63. Hasil ini didapatkan pada iterasi 4000 dan dengan ambang batas 0,9. Hal ini terjadi karena hanya 5 dari 34 *posting* dengan topik *Sort* adalah tentang *Bubble Sort.*

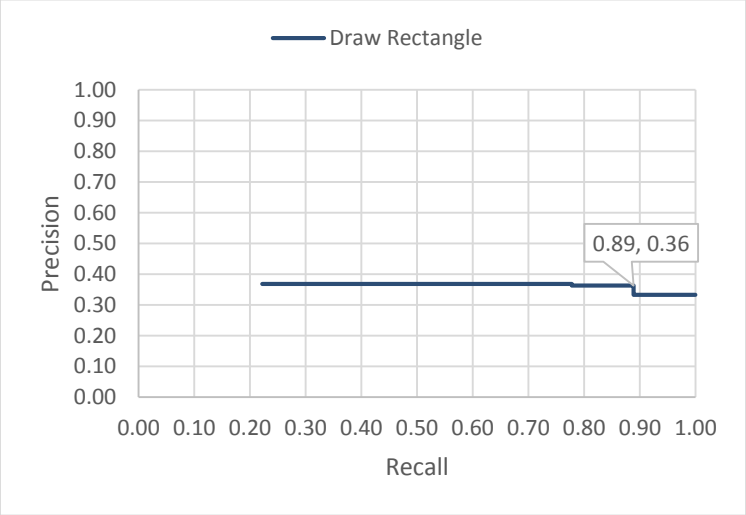
4.4.4 Pengujian dengan Pertanyaan *Draw Rectangle*

Pengujian selanjutnya adalah dengan menggunakan pertanyaan *draw rectangle.* Maksud dari pertanyaan tersebut adalah ingin mengetahui contoh kode untuk implementasi membuat gambar kotak menggunakan *Graph* pada Java. Pengujian dilakukan pada model topik yang bervariasi (1000, 2000, 4000, 8000) dan pada tiap iterasi dilakukan pengaturan ambang batas (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9). Gambar 4.8 menunjukkan salah satu percobaan menggunakan pertanyaan *draw rectangle.*



Gambar 4. 12 Percobaan dengan Pertanyaan *Draw Rectangle*.

Hasil percobaan dengan pertanyaan *draw rectangle* ditunjukkan pada Gambar 4.9. yaitu plot kurva interpolasi *precision & recall* untuk pertanyaan *draw rectangle*.

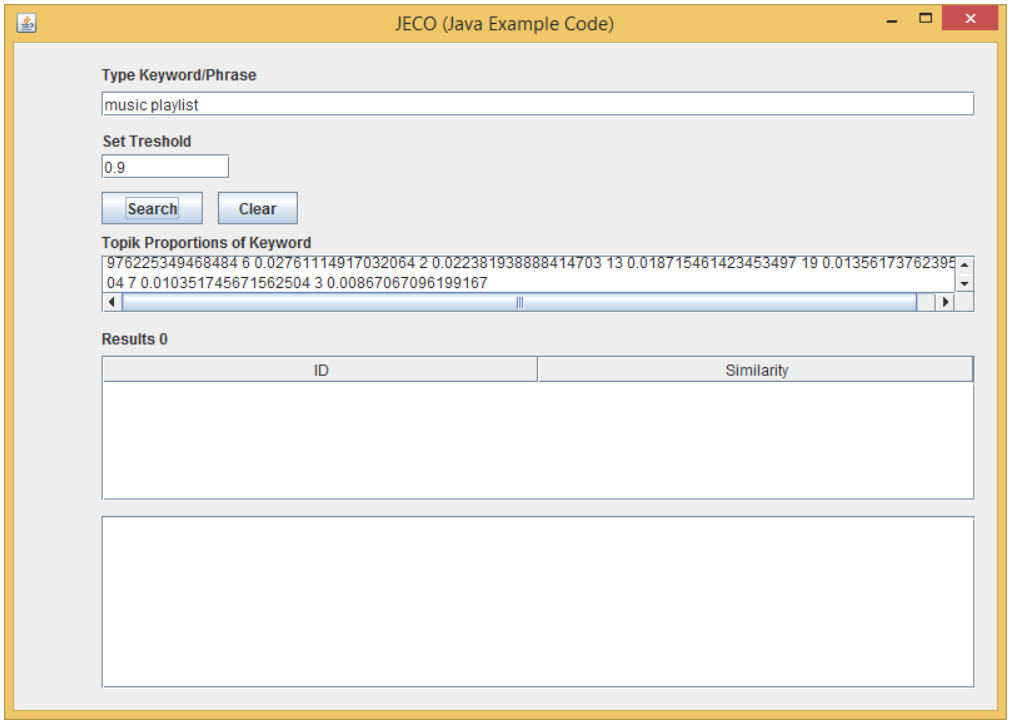


Gambar 4. 13 Plot Kurva Interpolasi *Precision & Recall* Hasil Percobaan dengan Pertanyaan *Draw Rectangle*.

Hasil percobaan pada pertanyaan kedua menunjukkan hasil yang kurang baik dimana nilai *precision* terbaik adalah 0,36 dengan *recall* 0,89. Hasil ini didapatkan pada iterasi 2000 dan dengan ambang batas 0,6. Pada percobaan ini banyak posting yang tidak relevan namun direkomendasikan oleh sistem. Ada 8 dari 32 *posting* dengan topik *Graphic* adalah tentang *draw rectangle*, namun karena kata *draw* muncul pada 24 *posting* dari 32 sehingga hal ini menyebabkan banyaknya *posting* yang tidak relevan yang direkomendasikan.

4.4.5 Pengujian dengan Pertanyaan *Music Playlist*

Pengujian kelima dilakukan dengan pertanyaan *music playlist.* Pertanyaan ini bertujuan menampilkan bagaimana kode untuk menambahkan musik ke dalam daftar antrian. Pertanyaan ini tidak relevan dengan topik yang ada pada dataset. Pengujian dilakukan pada model topik yang bervariasi (1000, 2000, 4000, 8000) dan pada tiap iterasi dilakukan pengaturan ambang batas (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9). Gambar 4.14 menunjukkan salah satu percobaan menggunakan pertanyaan *music playlist.*

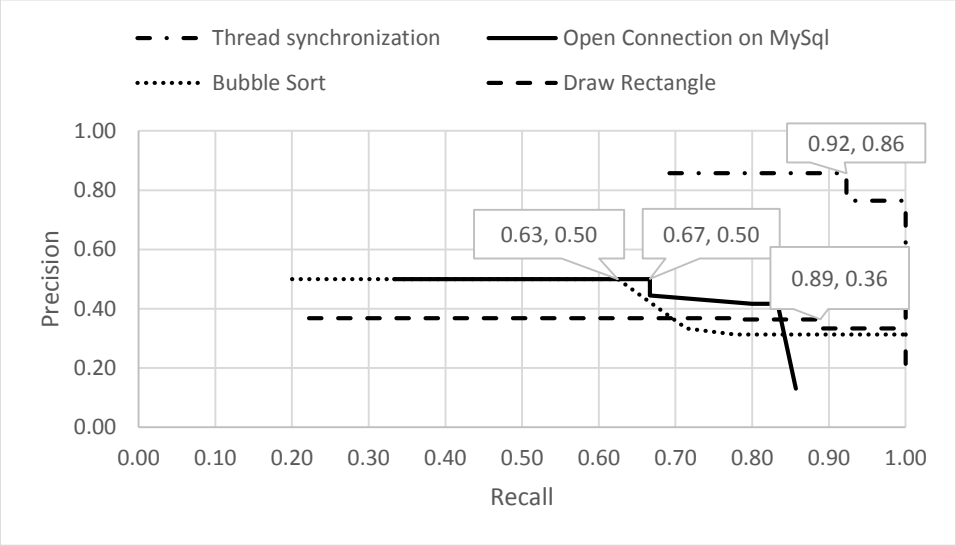


Gambar 4. 14 Percobaan dengan Pertanyaan *Music Playlist*.

Hasil percobaan menggunakan pertanyaan kelima menunjukkan bahwa sistem tidak merekomendasikan kode sumber.

4.4.6 Hasil Keseluruhan Pengujian

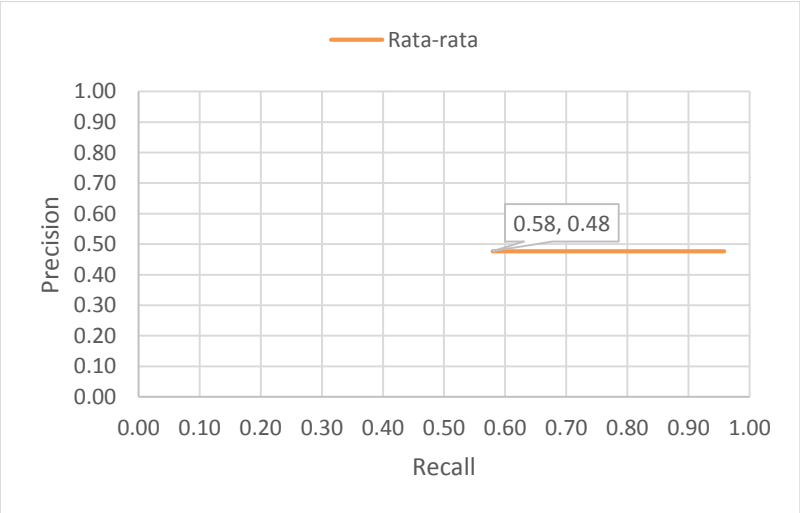
Seluruh hasil pengujian dari pertanyaan pertama sampai keempat ditunjukkan pada gambar 4.15. Dari keseluruhan plot kurva tersebut hanya hasil dari pertanyaan kedua(*open connection on mysql)* yang nilai *recall* nya tidak mencapai satu, hal ini terjadi kemungkinan disebabkan oleh jumlah *posting* yang relevan hanya lima dari 34 *posting* dan juga terdapat beberapa *posting* yang mirip dengan pertanyaan kedua yaitu *open connection on mssql.* Sehingga sistem kurang sensitif terhadap perbedaan antara *mysql* dan *mssql.*



Gambar 4. 15 Plot Kurva Interpolasi Precision & Recall Keseluruhan Hasil Percobaan.

4.5 Hasil Rerata Pengujian

Dari serangkaian pengujian yang telah dilakukan, didapatkan rata-rata tingkat *precision & recall.* Rata-rata ini hanya berasal dari hasil perhitungan *precision & recall* dari pertanyaan satu sampai dengan empat, sedangkan pertanyaan kelima tidak termasuk karena nilai yang selalu nol dimana jika dimasukkan akan menurunkan rata-rata. Gambar 4.15. yaitu plot kurva interpolasi rata-rata *precision & recall.*



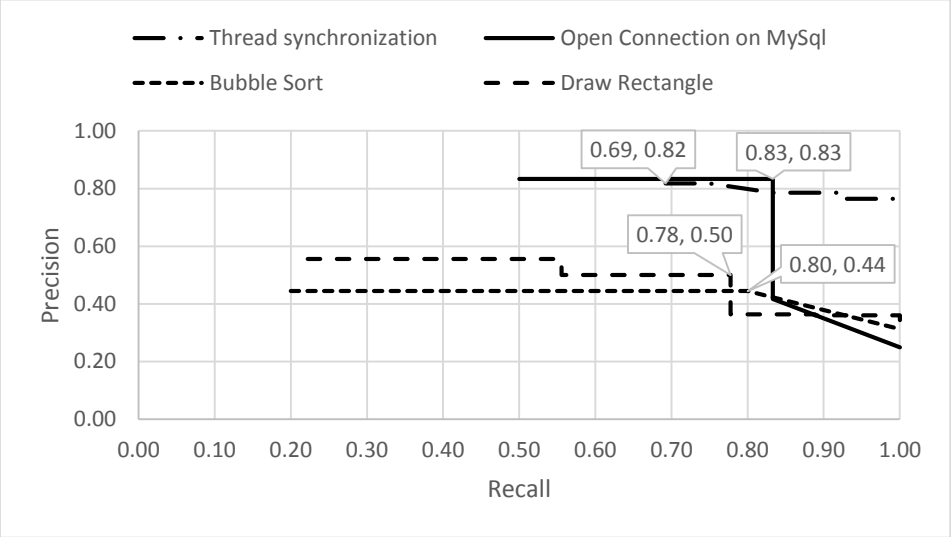
Gambar 4. 16 Plot Kurva Interpolasi Rata-rata *Precision & Recall*.

Hasil rata-rata terbaik yaitu nilai *precision* terbaik adalah 0,48 dengan *recall* 0,58. Hasil ini didapatkan pada iterasi ke-4000 dengan ambang batas 0.9. Hasil tersebut memang tidak terlalu memuaskan (hanya ±50%), hal ini disebabkan oleh kecilnya hasil *precision (0.22)* dan *recall (0.44)* pada pertanyaan keempat (*draw rectangle)*. Hasil yang didapatkan pada pertanyaan keempat dipengaruhi oleh jumlah kemunculan *draw rectangle* yaitu 8 posting dari 34 posting.

Hasil yang paling baik terlihat pada hasil dari pertanyaan pertama dimana *precision* sebagian besar berada pada bentang antara 0.50-0.86 dan *recall* berada pada 0.9-1.0. Hal ini disebabkan jumlah prosentase relevansi jawaban dengan dokumen cukup besar (70 %). Sementara peringkat kedua ditempati pertanyaan kedua(*open connection on mysql)* dengan tingkat *precision* berada pada level 0.2-0.5, dimana sebaran terbesar pada area ±0.4 dengan *recall* 0.6-1.0. Hal ini dipengaruhi oleh jumlah posting yang relevan dengan pertanyaan kedua(±20%). Peringkat ketiga ditempati pertanyaan ketiga(*bubble sort)* dengan tingkat *precision* pada level 0.1-0.5 dengan sebaran terbesar pada area ±0.2. Hal ini dipengaruhi oleh jumlah posting yang relevan hanya berkisar 15%.

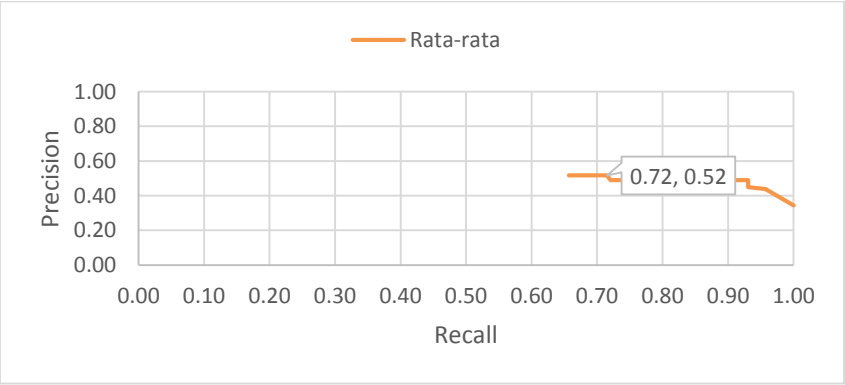
4.6 Pengujian Sensitivitas Sistem

Pengujian ini dilakukan untuk mengetahui bagaimana cara untuk meningkatkan sensitivitas sistem, yaitu mencari indikasi faktor apa yang dapat meningkatkan *precision/recall*. Dataset kemudian dipilih hanya terbatas pada kode sumber dan komentar saja yang terdapat baik pada pertanyaan maupun jawaban bukan keseluruhan dokumen. Hal ini bertujuan untuk mengetahui apakah kode dan komentar merupakan faktor yang dapat meningkatkan *precision/recall* atau tidak. Dataset kemudian dipraproses yaitu ditokenisasi, pemisahan nama identifier yang memenuhi aturan *camelCase* maupun *underscore*, eliminasi *stopword*, dan terakhir *stemming*. Model topik yang baru kemudian dibuat kembali. Proses pengujian juga dilakukan mulai dari pertanyaan satu sampai lima. Gambar 4.17 merupakan hasil uji sensitivitas sistem dengan data kode sumber dan komentar saja.



Gambar 4. 17 Plot Kurva Interpolasi Precision & Recall Hasil Uji Sensitivitas.

Hasil pengujian sensitivitas menunjukkan perubahan urutan ranking *precision/recall*. Pertanyaan ketiga(*open connection on mysql)* mengalami peningkatan, dimana awalnya *precision* dari 0.57 menjadi 0.83 dan *recall* dari 0.67 menjadi 0.83. Sedangkan pertanyaan kedua(*bubble sort)* menurun dari *precision 0.50* menjadi 0.44 dan *recall* dari 0.63 menjadi 0.80. Namun secara rata-rata, tingkat *precision & recall* dapat meningkat ± 4 % dimana sebelumnya nilai *precision 0.48* menjadi 0.52 dan *recall* dari 0.58 menjadi 0.72. Gambar 4.18 menunjukkan hasil rata-rata yang meningkat bila dibandingkan rata-rata jika keseluruhan dokumen diproses.

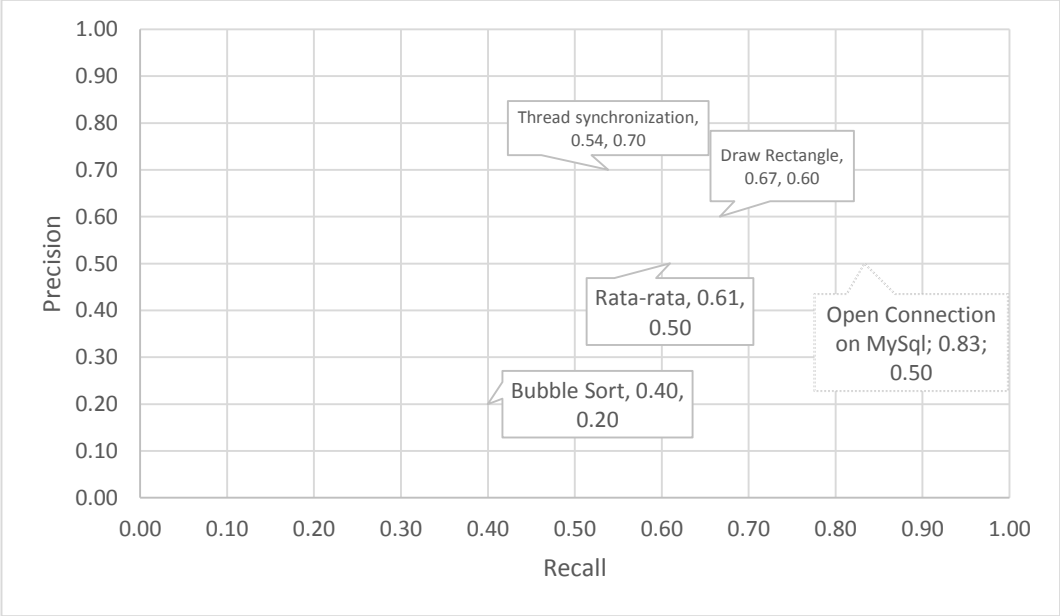


Gambar 4. 18 Plot Kurva Interpolasi Rerata Precision & Recall Hasil Uji Sensitivitas.

4.7 Perbandingan dengan Metode TF-IDF (*ExampleOverflow*)

Pengujian ini dilakukan untuk membandingkan hasil *precision* & *recall* antara metode pencarian dengan lokasi konsep dengan metode TF-IDF(*Term Frequency-Inverse Document Frequency*). Untuk mengimplementasikan TF-IDF maka dipilih Solr yaitu aplikasi *opensource* yang mampu melakukan pencarian data pada dokumen dengan algoritma TF-IDF didalamnya. Aplikasi ini juga digunakan oleh peneliti sebelumnya dalam mencari kode sumber di ExampleOverflow (Zagalsky, 2012). Penelitian ini mengadopsi sebagian metode yang digunakan pada ExampleOverflow (Zagalsky, 2012) yaitu memberi bobot empat pada judul, empat pada *tags*, satu pada jawaban, satu pada pertanyaan, dan dua pada kode tanpa dikalikan dengan *field* Score sosial. Alasan mengapa memilih metode yang digunakan ExampleOverflow dan bukan Stackoverflow oleh karena Stackoverflow tidak membuka algoritma pencarian apa yang dipakai pada portal mereka.

Pengujian dilakukan dengan memberikan 5 pertanyaan yang sama dengan pertanyaan diatas pada aplikasi Solr, untuk kemudian dilihat berapa tingkat *precision & recall* dari metode TF-IDF tersebut. Hasil pengujian dapat dilihat pada tabel berikut.



Gambar 4. 19 Hasil pengujian metode TF-IDF dengan Empat Pertanyaan.

Hasil pengujian dengan pertanyaan pertama (*thread synchronization*) Solr mampu merekomendasikan dengan tingkat *precision* sebesar 0.70 dan *recall* sebesar *0.*54. Pertanyaan kedua (*open connection on mysql*) Solr merekomendasikan kode dengan tingkat *precision* sebesar 0.50 dan *recall* sebesar *0.*83. Pertanyaan ketiga (*bubble sort)* Solr merekomendasikan kode dengan tingkat *precision* sebesar 0.20 dan *recall* sebesar *0.*40. Pertanyaan keempat (*draw rectangle)* Solr merekomendasikan kode dengan tingkat *precision* sebesar 0.60 dan *recall* sebesar *0.*67.

Secara rata-rata Solr mampu merekomendasikan kode dengan tingkat *precision* sebesar 0.50 dan *recall* sebesar *0.*61. Sedangkan penelitian ini mampu merekomendasikan kode dengan tingkat *precision* 0.52 dan *recall* 0.72. Hal ini berarti menunjukkan penelitian ini lebih unggul 2% dalam tingkat *precision* dan 11% dalam tingkat *recall*. Hal ini disebabkan karena pada penelitian ini dilakukan pemisahan identifier yang memenuhi aturan *camelCase/Underscore* sedangkan menggunakan pendekatan TF-IDF tidak memisahkan identifier.

4.7.1 Ancaman Validitas Internal

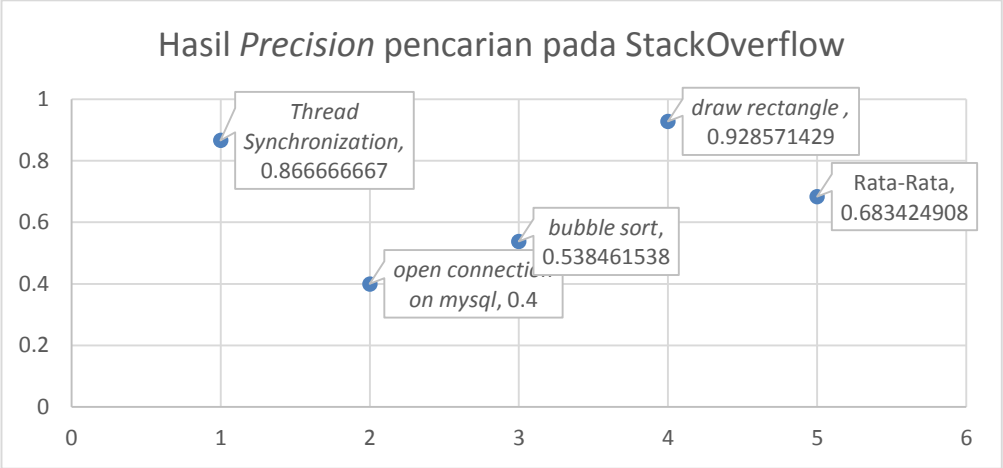
Untuk perbandingan pengujian dengan metode tf-idf pada penelitian ini mengadopsi sebagian metode yang digunakan pada ExampleOverflow (Zagalsky, 2012) yaitu memberi bobot empat pada judul, empat pada *tags*, satu pada jawaban, satu pada pertanyaan, dan dua pada kode tanpa dikalikan dengan *field* Score sosial. Hal ini dilakukan karena pada pengaturan Solr berbasis GUI tidak memungkinkan untuk mengaplikasikan rumus pemberian skor yang sama persis dengan yang dilakukan pada ExampleOverflow (Zagalsky, 2012). Oleh karena itu hasil keakuratan pencarian pada percobaan perbandingan ini kemungkinan tidak akan sama persis hasilnya dengan ExampleOverflow.

4.8 Perbandingan dengan Hasil Pencarian pada StackOverflow

Pengujian ini dilakukan untuk membandingkan hasil *precision* antara metode pencarian dengan lokasi konsep pada penelitian ini dengan hasil pada StackOverflow. *Precision* dipilih karena peneliti tidak dapat mengetahui berapa

data yang relevan pada StackOverflow. Hasil yang dipilih juga sebatas halaman pertama (15 dokumen), karena berapapun hasil yang dipilih tingkat *recall* juga akan meningkat seiring peningkatan jumlah hasil.

Pengujian dilakukan dengan memberikan pertanyaan yang sama dengan pertanyaan satu sampai empat pada percobaan pada fase pengujian(*thread synchronization, open connection on mysql, bubble sort, draw rectangle graphics).* Pertanyaan keempat berbeda dengan percobaan saat fase pengujian karena maksud pertanyaan keempat adalah bagaimana implementasi *draw rectangle* menggunakan *graphics.* Hasil pengujian dapat dilihat pada tabel berikut.



Gambar 4. 20 Hasil Pengujian Tingkat *Precision* Empat Pertanyaan pada StackOverflow.

Hasil pengujian dengan pertanyaan pertama (*thread synchronization*) StackOverflow mampu merekomendasikan dengan tingkat *precision* sebesar 0.86. StackOverflow merekomendasikan banyak dokumen dimana sebagian besar jawaban dari pertanyaan sudah diverifikasi oleh pemberi pertanyaan.

Hasil pengujian pertanyaan kedua (*open connection on mysql*), StackOverflow merekomendasikan kode dengan tingkat *precision* sebesar 0.4. Hasil ini paling kecil, StackOverflow merekomendasikan banyak dokumen yang tidak relevan karena rekomendasi yang diberikan pengatuan koneksi pada *framework* tertentu(spring, hibernate) dimana isinya hanya file xml. Selain itu sebagian dokumen berisi koneksi untuk database selain MySql.

Hasil pertanyaan ketiga (*bubble sort)* StackOverflow merekomendasikan kode dengan tingkat *precision* sebesar 0.53. Rekomendasi pada pertanyaan ini lebih banyak pertanyaan yang jawabannya bukan kode melainkan saran-saran saja atau kodenya minim. Selain itu juga banyak pertanyaan yang tidak diverifikasi jawabannya oleh pemberi pertanyaan.

Pertanyaan keempat (*draw rectangle graphics)* StackOverflow merekomendasikan kode dengan tingkat *precision* sebesar 0.92. Untuk pertanyaan ini, StackOverflow menghasilkan rekomendasi paling banyak. *Posting* tentang *draw rectangle graphics* hampir semua lengkap dengan kode dan sebagian besar sudah diverifikasi oleh pemberi pertanyaan.

Secara rata-rata hasil pencarian pada StackOverflow mampu merekomendasikan kode dengan tingkat *precision* sebesar 0.68, sedangkan penelitian ini mampu merekomendasikan kode dengan tingkat *precision* 0.52. Hal ini berarti menunjukkan penelitian ini kalah 16% dalam tingkat *precision* bila dibandingkan dengan StackOverflow (untuk 15 dokumen dengan skor tertinggi rekomendasi StackOverflow). Penyebabnya kemungkinan algoritma yang diterapkan StackOverflow lebih canggih dan penelitian ini perlu ditingkatkan metodenya, data maupun pra prosesnya.

BAB 5
KESIMPULAN & SARAN

5.1 Kesimpulan

Dari serangkaian percobaan yang telah telah dilakukan, dapat ditarik kesimpulan sebagai berikut.

1. Metode yang diajukan dalam penelitian ini mampu merekomendasikan kode sumber yang ada pada StackOverflow. Hasil terbaik yang dapat dicapai adalah pada tingkat *precision* 86% dan *recall* 92%. Sedangkan untuk rata-ratanya adalah *precision* 52% dan *recall* 72%.
2. Jumlah iterasi yang cocok dengan dataset penelitian ini adalah 2000 dan 4000 dimana hasil terbaik dan rata-rata terbaik hanya ada pada kedua iterasi tersebut.
3. Jumlah prosentase *posting* relevan mempengaruhi tingkat *precision & recall*, dimana semakin banyak prosentase dokumen yang relevan dengan pertanyaan maka akan semakin besar tingkat *precision* dan *recall.*
4. Dataset menggunakan kode & komentar dapat meningkatkan sensitivitas pencarian kode sumber bila dibandingkan dengan dataset pertanyaan, jawaban, kode dan komentar.
5. Bila dibandingkan dengan metode TF-IDF, penelitian ini lebih unggul 2% dalam tingkat *precision* dan 11% dalam tingkat *recall* oleh karena adanya proses pemisahan *identifier* yang memenuhi syarat *camelCase/underscore.*
6. Bila dibandingkan dengan hasil pencarian pada StackOverflow, penelitian ini kalah 16 % dalam tingkat *precision* untuk 15 dokumen dengan skor tertinggi pada mesin pencarian StackOverflow.

5.2 Saran

Berikut ini adalah beberapa saran untuk penelitian selanjutnya. Dalam mencari kode sumber pada StackOverflow dapat dikembangkan dengan

menggunakan pendekatan algoritma yang lain. Selain itu, pra proses yang lain seperti eliminasi kata yang terlalu sering muncul walaupun termasuk non *stopword* kemungkinan juga dapat meningkatkan capaian *precision & recall.* Media sosial yang lain seperti Facebook, Google+ kemungkinan juga merupakan situs yang potensial karena para pemrogram juga sering melakukan *posting* tutorial maupun kode sumber pada situs-situs tersebut.

DAFTAR PUSTAKA

Alexander Halavais, K. H. (2014). "Badges of Friendship: Social Influence and Badge Acquisition on Stack Overflow.". *System Sciences (HICSS), 2014 47th Hawaii International Conference on System Science* (pp. 1607-1615). Washington, DC, USA: IEEE Computer Society.

Allamanis, M. a. (2013). "Why, when, and what: analyzing stack overflow questions by topic, type, and code.". *Tenth International Workshop on Mining Software Repositories* (pp. 53-56). San Francisco, CA, USA: IEEE Press.

Arun, R. S. (2010). "On finding the natural number of topics with latent dirichlet allocation: Some observations.". *Advances in Knowledge Discovery and Data Mining, Part I* (pp. 391-402). Hyderabad, India: Springer Berlin Heidelberg.

Bela A. Frigyik, A. K. (2010). *Introduction to the Dirichlet Distribution and Related Processes.* Washington: UWEE.

Bogdan Dit, L. G. (2011). "Can Better Identifier Splitting Techniques Help Feature Location?". *The 2011 IEEE 19th International Conference on Program Comprehension (ICPC '11)* (pp. 11-20). Washington, DC: IEEE Computer Society.

Bogdan Vasilescu, A. S. (2014). "How social Q&A sites are changing knowledge sharing in open source software communities.". *The 17th ACM conference on Computer supported cooperative work & social computing* (pp. 342-354). Baltimore, Maryland, USA: ACM.

Christopher D. Manning, P. R. (2008). *Introduction to Information Retrieval.* New York: Cambridge University Press.

David M. Blei, A. Y. (2003). David M. Blei, Andrew Y. Ng, and Michael I. Jordan. *The Journal of Machine Learning Research*, 993-1022.

Erik Linstead, S. B. (2009). Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery*, 300-336.

Kartik Bajaj, K. P. (2014). "Mining questions asked by web developers.". *11th Working Conference on Mining Software Repositories (MSR 2014)* (pp. 112-121). Hyderabad, India: ACM.

Marcus, A. a. (2004). "An Information Retrieval Approach to Concept Location in Source Code". *Proceedings of the 11th Working Conference on Reverse Engineering* (pp. 214-223). Washington, DC, USA: IEEE Computer Society.

McCallum, A. K. (2002). *MALLET: A Machine Learning for Language Toolkit.* Retrieved 2014, from MALLET: A Machine Learning for Language Toolkit.: http://mallet.cs.umass.edu

Ming-Yang (Jerry) Lin, R. A. (2006). "A Java Reuse Repository for Eclipse using LSI.". *Australian Software Engineering Conference (ASWEC '06)* (pp. 351-362). Washington, DC, USA: IEEE Computer Society.

Ossher, J. a. (2009). "SourcererDB: An Aggregated Repository of Statically Analyzed and Cross-linked Open Source Java Projects". *The 2009 6th IEEE International Working Conference on Mining Software Repositories (MSR '09)* (pp. 183-186). Washington, DC: IEEE Computer Society.

Oxford. (2014). *www.oxforddictionaries.com/.* Retrieved from www.oxforddictionaries.com/: www.oxforddictionaries.com/definition/english/corpus?q=corpus

Savage, T. a. (2010). "TopicXP: Exploring Topics in Source Code Using Latent Dirichlet Allocation". *The 2010 IEEE International Conference on Software Maintenance* (pp. 1-6). Washington, DC: IEEE Computer Society.

Squire, M. a. (2014). "A Bit of Code: How the Stack Overflow Community Creates Quality Postings". *the 2014 47th Hawaii International Conference on System Sciences* (pp. 1425-1434). Washington, DC, USA: IEEE Computer Society.

StackOverflow. (2014). *http://stackoverflow.com/*. Retrieved from http://stackoverflow.com/: http://stackoverflow.com/

Steidl, D. H. (2013). "Quality analysis of source code comments.". *2013 21st International Conference on Program Comprehension (ICPC)* (pp. 83-92). IEEE.

Subramanian, S. a. (2013). "Making Sense of Online Code Snippets". *The 10th Working Conference on Mining Software Repositories* (pp. 85-88). San Francisco, CA, USA: IEEE Press.

Tausczik, Y. R. (2014). "Collaborative Problem Solving: A Study of MathOverflow". *17th ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW 2014)* (pp. 355-367). Baltimore, Maryland, USA: ACM.

Ted J. Biggerstaff, B. G. (1994). Program understanding and the concept assignment problem. *Commun. ACM*, 72-82.

Thomas, S. W. (2014). Studying Software Evolution Using Topic Models. *Science of Computer Programming*, 457-479.

Vinz, B. L. (2008). Improving Program Comprehension by Combining Code Understanding with Comment Understanding. *Knowledge-Based Systems*, 813-825.

Wang, S. a. (2013). "An Empirical Study on Developer Interactions in StackOverflow". *The 28th Annual ACM Symposium on Applied Computing* (pp. 1019-1024). Coimbra, Portugal: ACM.

Werner Janjic, O. H. (2013). "An unabridged source code dataset for research in software reuse.". *10th Working Conference on Mining Software Repositories (MSR '13)* (pp. 339-342). Piscataway, NJ, USA,: IEEE Press.

Wilde, N. a. (2003). A Comparison of Methods for Locating Features in Legacy Software. *Journal of Systems and Software*, 105-114.

Zagalsky, A. a. (2012). "Example Overflow: Using Social Media for Code Recommendation". *The third International Workshop on Recommendation Systems for Software Engineering* (pp. 38-42). Zurich, Switzerland: IEEE Press.

LAMPIRAN

Tabel Percobaan Keseluruan Dokumen

# Topics	Iterations	Treshold	Precission						RECALL					
			Quest. 1	Quest. 2	Quest. 3	Quest. 4	Quest. 5	Avg	Quest. 1	Quest. 2	Quest. 3	Quest. 4	Quest. 5	Avg
			Thread Synchronization	open connection on mysql	buble sort	draw rectangle	music playlist		Thread Synchronization	open connection on mysql	buble sort	draw rectangle	music playlist	
20	1000	0.1	0.18	0.13	0.06	0.17	0.00	0.14	1.00	0.86	1.00	0.91	0.00	0.94
20	1000	0.2	0.57	0.33	0.12	0.29	0.00	0.33	1.00	0.67	1.00	0.90	0.00	0.89
20	1000	0.3	0.60	0.40	0.18	0.30	0.00	0.37	1.00	0.67	1.00	0.90	0.00	0.89
20	1000	0.4	0.60	0.44	0.17	0.28	0.00	0.37	1.00	0.67	0.80	0.80	0.00	0.82
20	1000	0.5	0.60	0.33	0.17	0.29	0.00	0.35	0.92	0.33	0.80	0.80	0.00	0.71
20	1000	0.6	0.60	0.40	0.15	0.33	0.00	0.37	0.92	0.33	0.60	0.80	0.00	0.66
20	1000	0.7	0.60	0.40	0.17	0.32	0.00	0.37	0.92	0.33	0.60	0.70	0.00	0.64
20	1000	0.8	0.63	0.40	0.22	0.35	0.00	0.40	0.92	0.33	0.40	0.70	0.00	0.59
20	1000	0.9	0.64	0.40	0.22	0.37	0.00	0.41	0.75	0.33	0.40	0.70	0.00	0.55
20	2000 (2)	0.1	0.50	0.25	0.15	0.26	0.00	0.29	0.93	0.83	1.00	0.89	0.00	0.91
20	2000 (2)	0.2	0.68	0.36	0.17	0.31	0.00	0.38	1.00	0.83	1.00	0.89	0.00	0.93
20	2000 (2)	0.3	0.68	0.38	0.19	0.31	0.00	0.39	1.00	0.83	1.00	0.89	0.00	0.93
20	2000 (2)	0.4	0.68	0.42	0.19	0.32	0.00	0.40	1.00	0.83	0.89	0.80	0.00	0.88
20	2000 (2)	0.5	0.68	0.27	0.20	0.32	0.00	0.37	1.00	0.50	0.89	0.80	0.00	0.80
20	2000 (2)	0.6	0.71	0.27	0.20	0.36	0.00	0.39	0.92	0.50	0.80	0.80	0.00	0.76
20	2000 (2)	0.7	0.75	0.38	0.17	0.37	0.00	0.42	0.92	0.50	0.78	0.70	0.00	0.73
20	2000 (2)	0.8	0.86	0.38	0.21	0.33	0.00	0.44	0.92	0.50	0.57	0.67	0.00	0.67
20	2000 (2)	0.9	0.85	0.38	0.25	0.33	0.00	0.45	0.85	0.50	0.50	0.67	0.00	0.63
20	2000	0.1	0.50	0.25	0.15	0.26	0.00	0.29	1.00	0.83	1.00	0.89	0.00	0.93
20	2000	0.2	0.68	0.36	0.17	0.31	0.00	0.38	1.00	0.83	1.00	0.89	0.00	0.93
20	2000	0.3	0.68	0.31	0.19	0.31	0.00	0.37	1.00	0.80	1.00	0.89	0.00	0.92
20	2000	0.4	0.68	0.33	0.19	0.32	0.00	0.38	1.00	0.80	1.00	0.89	0.00	0.92
20	2000	0.5	0.68	0.27	0.20	0.32	0.00	0.37	1.00	0.60	1.00	0.89	0.00	0.87
20	2000	0.6	0.71	0.27	0.20	0.36	0.00	0.39	0.92	0.60	1.00	0.89	0.00	0.85
20	2000	0.7	0.75	0.38	0.17	0.37	0.00	0.42	0.92	0.60	0.80	0.78	0.00	0.78
20	2000	0.8	0.86	0.38	0.21	0.33	0.00	0.44	0.92	0.50	0.80	0.67	0.00	0.72
20	2000	0.9	0.85	0.38	0.25	0.33	0.00	0.45	0.85	0.50	0.43	0.67	0.00	0.61
20	4000	0.1	0.50	0.31	0.19	0.23	0.00	0.31	1.00	0.80	1.00	1.00	0.00	0.95
20	4000	0.2	0.65	0.44	0.26	0.27	0.00	0.41	1.00	0.67	1.00	1.00	0.00	0.92
20	4000	0.3	0.65	0.50	0.26	0.28	0.00	0.42	1.00	0.67	1.00	1.00	0.00	0.92
20	4000	0.4	0.65	0.43	0.28	0.27	0.00	0.41	1.00	0.60	0.83	0.89	0.00	0.83
20	4000	0.5	0.65	0.33	0.28	0.27	0.00	0.38	1.00	0.33	0.83	0.78	0.00	0.74
20	4000	0.6	0.68	0.40	0.29	0.24	0.00	0.40	1.00	0.33	0.71	0.60	0.00	0.66
20	4000	0.7	0.71	0.40	0.33	0.23	0.00	0.42	0.92	0.33	0.71	0.56	0.00	0.63
20	4000	0.8	0.75	0.40	0.38	0.23	0.00	0.44	0.92	0.33	0.63	0.56	0.00	0.61
20	4000	0.9	0.79	0.40	0.50	0.22	0.00	0.48	0.92	0.33	0.63	0.44	0.00	0.58
20	4000(2)	0.1	0.50	0.31	0.19	0.23	0.00	0.31	1.00	0.80	1.00	1.00	0.00	0.95
20	4000(2)	0.2	0.65	0.44	0.26	0.27	0.00	0.41	1.00	0.67	1.00	1.00	0.00	0.92
20	4000(2)	0.3	0.65	0.50	0.26	0.28	0.00	0.42	1.00	0.67	1.00	1.00	0.00	0.92
20	4000(2)	0.4	0.65	0.43	0.28	0.27	0.00	0.41	1.00	0.60	0.83	0.80	0.00	0.81
20	4000(2)	0.5	0.65	0.33	0.28	0.27	0.00	0.38	1.00	0.33	0.83	0.78	0.00	0.74
20	4000(2)	0.6	0.68	0.40	0.29	0.24	0.00	0.40	1.00	0.33	0.71	0.67	0.00	0.68
20	4000(2)	0.7	0.71	0.40	0.33	0.23	0.00	0.42	0.92	0.33	0.71	0.56	0.00	0.63
20	4000(2)	0.8	0.75	0.40	0.38	0.23	0.00	0.44	0.92	0.33	0.63	0.56	0.00	0.61
20	4000(2)	0.9	0.79	0.40	0.50	0.22	0.00	0.48	0.92	0.33	0.63	0.44	0.00	0.58
20	8000	0.1	0.38	0.36	0.16	0.24	0.00	0.29	1.00	0.83	1.00	1.00	0.00	0.96
20	8000	0.2	0.68	0.42	0.26	0.31	0.00	0.42	1.00	0.83	1.00	1.00	0.00	0.96
20	8000	0.3	0.68	0.42	0.28	0.31	0.00	0.42	1.00	0.83	1.00	1.00	0.00	0.96
20	8000	0.4	0.68	0.42	0.31	0.33	0.00	0.44	1.00	0.83	1.00	1.00	0.00	0.96
20	8000	0.5	0.68	0.40	0.31	0.32	0.00	0.43	1.00	0.67	1.00	0.89	0.00	0.89
20	8000	0.6	0.72	0.40	0.31	0.30	0.00	0.43	1.00	0.67	1.00	0.78	0.00	0.86
20	8000	0.7	0.76	0.44	0.27	0.30	0.00	0.44	1.00	0.67	0.80	0.67	0.00	0.78
20	8000	0.8	0.75	0.38	0.29	0.28	0.00	0.42	0.92	0.50	0.80	0.56	0.00	0.69
20	8000	0.9	0.75	0.38	0.13	0.14	0.00	0.35	0.69	0.50	0.20	0.22	0.00	0.40

Tabel Percobaan Kode Sumber & Komentar Saja

# Topics	Iterations	Treshold	Precission						RECALL					
			Quest. 1	Quest. 2	Quest. 3	Quest. 4	Quest. 5	Avg	Quest. 1	Quest. 2	Quest. 3	Quest. 4	Quest. 5	Avg
			Thread Synchronization	open connection on mysql	bubble sort	draw rectangle	music playlist		Thread Synchronization	open connection on mysql	bubble sort	draw rectangle	music playlist	
20	1000	0.1	0.21	0.25	0.08	0.15	0.00	0.17	1.00	1.00	1.00	1.00	0.00	1.00
20	1000	0.2	0.57	0.33	0.17	0.30	0.00	0.34	1.00	0.83	1.00	1.00	0.00	0.96
20	1000	0.3	0.60	0.38	0.14	0.31	0.00	0.36	1.00	0.83	1.00	1.00	0.00	0.96
20	1000	0.4	0.60	0.42	0.21	0.33	0.00	0.39	1.00	0.83	1.00	1.00	0.00	0.96
20	1000	0.5	0.63	0.42	0.22	0.35	0.00	0.40	1.00	0.83	1.00	1.00	0.00	0.96
20	1000	0.6	0.63	0.50	0.23	0.35	0.00	0.43	1.00	0.83	1.00	1.00	0.00	0.96
20	1000	0.7	0.73	0.44	0.21	0.36	0.00	0.44	0.92	0.67	0.80	1.00	0.00	0.85
20	1000	0.8	0.73	0.44	0.18	0.35	0.00	0.43	0.92	0.67	0.60	0.78	0.00	0.74
20	1000	0.9	0.79	0.44	0.21	0.27	0.00	0.43	0.92	0.67	0.60	0.44	0.00	0.66
20	2000 (2)	0.1	0.46	0.83	0.15	0.30	0.00	0.43	1.00	0.83	1.00	0.89	0.00	0.93
20	2000 (2)	0.2	0.63	0.83	0.17	0.31	0.00	0.49	1.00	0.83	1.00	0.89	0.00	0.93
20	2000 (2)	0.3	0.63	0.83	0.19	0.31	0.00	0.49	1.00	0.83	1.00	0.89	0.00	0.93
20	2000 (2)	0.4	0.63	0.67	0.19	0.31	0.00	0.45	1.00	0.67	1.00	0.89	0.00	0.89
20	2000 (2)	0.5	0.63	0.50	0.20	0.32	0.00	0.41	1.00	0.50	1.00	0.89	0.00	0.85
20	2000 (2)	0.6	0.65	0.50	0.20	0.36	0.00	0.43	0.92	0.50	1.00	0.89	0.00	0.83
20	2000 (2)	0.7	0.69	0.50	0.17	0.37	0.00	0.43	0.92	0.50	0.80	0.78	0.00	0.75
20	2000 (2)	0.8	0.79	0.50	0.21	0.33	0.00	0.46	0.92	0.50	0.80	0.67	0.00	0.72
20	2000 (2)	0.9	0.77	0.50	0.25	0.33	0.00	0.46	0.83	0.50	0.60	0.67	0.00	0.65
20	4000(2)	0.1	0.41	0.36	0.06	0.28	0.00	0.28	1.00	0.83	1.00	0.78	0.00	0.90
20	4000(2)	0.2	0.57	0.36	0.12	0.50	0.00	0.39	1.00	0.83	1.00	0.78	0.00	0.90
20	4000(2)	0.3	0.57	0.36	0.18	0.50	0.00	0.40	1.00	0.83	1.00	0.67	0.00	0.88
20	4000(2)	0.4	0.57	0.42	0.19	0.50	0.00	0.42	1.00	0.83	1.00	0.56	0.00	0.85
20	4000(2)	0.5	0.57	0.36	0.21	0.50	0.00	0.41	1.00	0.67	1.00	0.56	0.00	0.81
20	4000(2)	0.6	0.57	0.36	0.22	0.56	0.00	0.43	1.00	0.67	1.00	0.56	0.00	0.81
20	4000(2)	0.7	0.60	0.38	0.26	0.56	0.00	0.45	1.00	0.50	1.00	0.56	0.00	0.76
20	4000(2)	0.8	0.69	0.38	0.31	0.50	0.00	0.47	0.92	0.50	1.00	0.44	0.00	0.72
20	4000(2)	0.9	0.82	0.38	0.44	0.43	0.00	0.52	0.75	0.50	0.80	0.33	0.00	0.60
20	8000	0.1	0.38	0.36	0.16	0.24	0.00	0.29	1.00	0.83	1.00	1.00	0.00	0.96
20	8000	0.2	0.68	0.42	0.26	0.31	0.00	0.42	1.00	0.83	1.00	1.00	0.00	0.96
20	8000	0.3	0.68	0.42	0.28	0.31	0.00	0.42	1.00	0.83	1.00	1.00	0.00	0.96
20	8000	0.4	0.68	0.42	0.31	0.33	0.00	0.44	1.00	0.83	1.00	1.00	0.00	0.96
20	8000	0.5	0.68	0.40	0.31	0.32	0.00	0.43	1.00	0.67	1.00	0.89	0.00	0.89
20	8000	0.6	0.72	0.40	0.31	0.30	0.00	0.43	1.00	0.67	1.00	0.78	0.00	0.86
20	8000	0.7	0.76	0.44	0.27	0.30	0.00	0.44	1.00	0.67	0.80	0.67	0.00	0.78
20	8000	0.8	0.75	0.38	0.29	0.28	0.00	0.42	0.92	0.50	0.80	0.56	0.00	0.69
20	8000	0.9	0.75	0.38	0.13	0.14	0.00	0.35	0.69	0.50	0.20	0.22	0.00	0.40

Tabel Percobaan dengan Metode TF-IDF

Precission						RECALL					
Quest. 1	Quest. 2	Quest. 3	Quest. 4	Quest. 5	Avg	Quest. 1	Quest. 2	Quest. 3	Quest. 4	Quest. 5	Avg
Thread Synchronization	open connection on mysql	bubble sort	draw rectangle	music playlist		Thread Synchronization	open connection on mysql	bubble sort	draw rectangle	music playlist	
0.70	0.50	0.20	0.60	0.00	0.50	0.54	0.83	0.40	0.67	0.00	0.61

Tabel Percobaan dengan Stackoverflow

Precission					
Quest. 1	Quest. 2	Quest. 3	Quest. 4	Quest. 5	Avg
Thread Synchronization	open connection on mysql	bubble sort	draw rectangle	music playlist	
0.87	0.40	0.54	0.86	0.00	0.67

BIOGRAFI PENULIS

Penulis lahir di Malang 31 tahun silam. Lulus S1 (S.Kom) pada 2006 penulis kemudian malang melintang bekerja dibidang IT. Mulai dari pemrogram lepas, teknisi, sampai terakhir penulis menjadi dosen di Universitas Brawijaya sejak 2008 hingga sekarang. Penulis kemudian menempuh S2 di jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya pada tahun 2012 serta kemudian lulus S2 (M.Kom) pada tahun 2015. Penulis menekuni bidang ilmu rekayasa perangkat lunak. Beberapa publikasi internasional sudah dihasilkan pada rentang waktu studi S2 tersebut. Hingga kini penulis masih menekuni bidang yang sama. Untuk berkorespondensi dengan penulis dapat melalui menghubungi melalui email arwan@ub.ac.id.